AD-A207 807

COMPUTER COMMAND AND CONTROL COMPANY

2401 WALNUT STREET, SUITE 402 • PHILADELPHIA, PA 19103 • (215) 854 0555

FINAL REPORT:

AN INTELLIGENT MATHEMATICAL MODELLING SYSTEM - MATHMODEL

March 1989
Submitted to:
Dr. A. Waksman (202-767-5027)
Air Force Office of Scientific Research (AFOSR/NM)
Directorate of Mathematical and Information Science
Building 410
Bolling Air Force Base, DC 20332-6448

Research sponsored by the Air Force Office of Scientific Research (AFSC), under Contract F49620-88-C-0116. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

DTIC
ELECTE
MAY 11 1989
S D
H

89 5 11 056

SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): Computer Command and Control Company	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 89-0530
6a. NAME OF PERFORMING ORGANIZATION: Computer Command & Control Co. / 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
6c. ADDRESS (City, State, and ZIP Code): 2401 Walnut Street, Ste. 402, Philadelphia, PA 19103	7b. ADDRESS (City, State, and ZIP Code): Building 410, Bolling AFB, DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Air Force Office of Scientific Research / 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-88-C-0116
8c. ADDRESS (City, State, and ZIP Code): Building 410, Bolling Air Force Base, DC 20322-6448	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 6110RF; PROJECT NO. 3005; TASK NO A1; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification): An Intelligent Mathematical Modelling System - MATHMODEL

12. PERSONAL AUTHOR(S): E. Lock, X. Ge, N. Prywes

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 8/1/88 TO 3/31/89	3/31/89	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Mathematical Modelling; Numerical Analysis; Specification Languages; Logical Checking; Symbolic Manipulation; Code Generation

19. This is the final report for SBIR grant number F49620-88-C-0116 from the Air Force Office of Scientific Research (AFOSR). It reports:

1. Enhancement of the MATHMODEL system developed at the University of Pennsylvania and its incorporation in Computer Command and Control Company's (CCCC) MODEL system.
2. Demonstration of the unique capabilities through examples.

Developing mathematical models of over 30 equations is reported to exceed $5,000 per equation. It also requires a high level of expertise from the developers for formulating the model, employing solution methods, testing it, and verifying its applicability. The MATHMODEL system represents a new and radical approach of automation of mathematical modelling with a potential for an order of magnitude reduction over current systems in cost and requisite expertise.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr A. Waksman	22b. TELEPHONE (Include Area Code): (202) 767-5027 / 22c. OFFICE SYMBOL: NM

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED

MATHMODEL is directed to the key problems in the **general application** of **large scale mathematical models:**

- *Organization and the comprehension* of large mathematical models,
- *Programming and integration of diverse solution methods*, and
- *Generality* and *extensibility*.

MATHMODEL has the following capabilities:

Language: a user can state piecewise assertions about the model in a very natural and general way. These assertions describe identities, structural relations or optimizations in the area to be modeled.

MATHMODEL simplifies the modelling process by:

- filling-in implicit details,
- checking completeness of the model,
- decomposing the model into interrelated subsets of assertions,
- partitioning assertions into inter-related subsets,
- mapping these sets into respective solution methods,
- manipulating assertions into representations needed for selected solution methods,
- generating efficient programs,
- evaluating the overall model, and
- reporting the solutions.

MATHMODEL integrates advances from a number of Artificial Intelligence related areas, i.e. specification languages, analysis of specification, symbolic manipulation, numerical analysis, and automatic generation and optimization of programs.

Following an introductory section, the three project tasks are described. The report also describes MATHMODEL's novel capabilites, how it works and how to use it. The report concludes with discussion of the marketplace for MATHMODEL and plans for continuing development in Phases II and III.

QUALITY INSPECTED 2

Accession For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist Special
A-1

Table of Contents

List of Figures

1. INTRODUCTION AND SUMMARY

1.1. Objectives

Mathematical models are used in a wide range of applications. They are used to explore social and physical systems. They have become essential in a great variety of design areas, from aircraft to urban centers. They are also used for planning by large private and public organizations. In many cases they are incorporated in operational systems in which they are used to make critical real-time decisions, based on dynamic information received from sensors. Finally, mathematical modelling has recently been used in expert systems. As the art of mathematical modelling advanced, these models have become more realistic and detailed, and as a result they have also grown greatly in size.

The development and use of mathematical models has been extremely costly in time and funds. Fromm [Fromm 75] has surveyed 650 mathematical models with a median size of 25 equations. Only 6 of these had more than 1000 equations. The cost per *equation* was on average 3 man-weeks. The cost per equation increases greatly as the number of equations in a model increases.

The cost problem has stimulated development of numerous mathematical modelling systems over the last two decades. There has been a conflict between generality and specialization in these systems. They have typically specialized in a particular application and/or solution method. Then, as the application changed with time, they could not be easily modified to respond to the new requirement.

The need for the new capabilities introduced in MATHMODEL has been expressed frequently in the past, as illustrated by the following quotation [Waren 87]:

We anticipate that a growing number of analysis and modelling systems of various kinds will provide optimization as an integral component. As the degree of integration of the modelling and optimization system improve, the ability of the unsophisticated user to employ nonlinear optimization will increase dramatically. This change will require additional developments in related areas such as the automatic detection of linearities and nonlinearities, automatic problem classification, and automatic selection of the best solution algorithm. We expect that all of these developments will be forthcoming and that artificial intelligence techniques may play an

important role.

Such a system, called MATHMODEL has been developed by X. Ge in his research at the University of Pennsylvania. It is a very complex and large multi-phase system. It consists of 142 modules and 60,000 lines of PL/1 code. MATHMODEL is based on an old (1984) version of the MODEL system, which automatically translates equational specifications into highly efficient programs in PL/1.

This is the Final Report of an SBIR Phase I project supported by the Air Force Office of Scientific Research under grant number F59620-88-C-0116. Computer Command and Control Company (CCCC) has a much more advanced and reliable version of MODEL that generates programs in several languages (PL/1, C and Ada) and that runs on several computers (IBM and Digital). It also generates programs that can be executed in parallel on distributed computers. Most important, CCCC's MODEL contains many more operations useful in mathematical modelling (e.g. matrix algebra, relational algebra, etc.). This version is much more reliable and robust and is well documented. The project has merged MATHMODEL's capabilities with those of CCCC's MODEL and has transformed MATHMODEL into a greatly more effective tool for mathematical modelling than any system developed to date (Task 1).

It has also demonstrated MATHMODEL's advantages through examples that show the ease and high productivity in using it (Task 2).

It has also identified the market for MATHMODEL and developed a strategy for its commercialization (Task 3).

To make MATHMODEL widely attractive, it will be necessary in Phase II to place MATHMODEL into an environment with the following capabilities:

1. Use of a powerful workstation,
2. Prototyping and reusability through an incorporated database of models,
3. Use of graphics for input of models,
4. Generation of programs for parallel processing,
5. Generation of programs in Fortran.

Also the market scope will be expanded to include mathematical modelling related activities, such as simulation and training. These capabilities will be a basis for a very powerful next

generation mathematical modelling system. It will serve in Phase III to attract necessary capitalization for commerical level offerring and support of MATHMODEL in Phase III.

1.2. Outline of the Report

The report consists of six sections. The presentation in the remaining five sections is briefly summarized below.

Section 2 Overview of the Capabilities of MATHMODEL. This section describes "what is MATHMODEL?" from the point of view of the prospective user.

Section 3 Task 1: Enhancement of MATHMODEL. This section describes "how MATHMODEL works" after the merger of the version developed by X. Ge with CCCC's MODEL

Section 4 Task 2: Demonstration of MATHMODEL Capabilities. This section describes "how MATHMODEL is used" in the course of three short examples. Larger examples could not be presented due to time and cost limitations. However, a related larger example is described.

Section 5 Task 3: Investigation of the Marketplace for MATHMODEL. This section describes "who are the prospective users of MATHMODEL".

Section 6 Conclusion. This section describes the technical environment for MATHMODEL that will be developed in Phase III: computers, operating systems, languages, databases, graphics, and their integration to provide an order of magnitude improved mathematical modelling system.

2. OVERVIEW OF THE NOVEL CAPABILITIES OF MATHMODEL

2.1. Overview

MATHMODEL has been directed to automating the core of the difficulties in *large scale mathematical modelling:*

1. The organization of large mathematical models as an aid to comprehension.
2. The integration of diverse solution methods.
3. Providing generality as well as ease in growth.

MATHMODEL takes over partitioning and organizing of the model. The organization

scheme is used to facilitate comprehension by the user. It also possesses intelligence to analyze assertions and select for them preferred solution methods. The system is *open-ended* for the purpose of enhancing it easily with new approaches and solution methods for them.

The innovations in MATHMODEL are illustrated by the following capabilities.

- Describing a model in terms of assertions: equations, optimizations and variables.
- Filling-in implicitly expressed details of data and assertions in the model,
- Checking completeness of the model,
- Partitioning the model's assertions into interrelated subsets,
- Mapping these sets into respective solution methods,
- Manipulating assertions into representations needed for selected solution methods,
- Generating efficient programs for the model's procedures,
- Testing and evaluating the overall model, and
- Reporting the results of the ensuing computation.

MATHMODEL incorporates advances in a number of areas of Artificial Intelligence to make feasible the generalized mathematical modelling system that can perform organization and manipulation tasks and easily grow in its capabilities. These areas include

- Specification languages,
- Logical analysis of the specifications,
- Symbolic manipulation of assertions,
- Numerical analysis,
- Automatic generation and optimization of programs.

2.2. The Intelligent Capabilities of MATHMODEL

The intelligent capabilities are classified below into five areas. They are summarized in figure 2.1.

2.2.1. Specification Language

Mathematical modelling languages have, in the past, been influenced by the formalism used in solution methods [Dolk 86]. Instead, MATHMODEL uses the recent advances in the area of *specification languages* [Prywes 75] [Backus 78] [Zave 85] [Sterli 86] to provide a simple general purpose language that employs commonly used mathematics terminology and semantics. The specification language has many advantages. It is independent of any computer implementation. The main idea is that the user composes a set of assertions that are considered as axioms in the environment being modeled. The semantics of this language are the same as those used in mathematics -- to find a solution (values of variables) for which all the assertions are true. The language also includes declarations of variables and their structures. To be close to mathematical modelling, the assertions use the syntax of regular or Boolean algebra's equalities or inequalities (differential equations must be transformed into difference equations). The same language is used solely for all maintenance and documentation of the mathematical model; the user would not even need to know the programming language that is used in the implementation of the computations.

```
Language     -- Unrestricted form of equalities and inequalities,
             -- Mix of array variables of different dimensionality
                and dynamic sizes of dimensions (mixed shapes),
             -- Arbitrary order of statements,
             -- Generalized data bases and reports.

Analysis     -- Tolerance of omissions. Automatic fill-in of:
                      declarations of data,
                      subscripts,
                      equations,
                      selection of numerical solution methods.
             -- Checking:
                      completeness,
                      dimensionality of arrays,
                      sizes of the array dimensions,
                      data types,
                      circularity of definitions.

             -- Organization:
                      finding causality,
                      identifying groups of statements that are
                          interdependent and must be solved
                          simultaneously,
                      selecting suitable solution methods,
                      combining automatically different solution
                          methods,

Symbolic     -- Manipulating the user's form into a form
Manipulation       required by the solution method.

Automatic    -- Avoiding the need to compose and test procedural
Programming        programs,
             -- Generating a highly efficient program for
                   solving the problem given by the specification
                   (the generated program is reusable),
             -- Prototyping,
             -- Parallel processing.

Numerical    -- Built-in six key methods for solving simultaneous
Analysis        equations and optimization, linear or nonlinear,
             -- Solution methods for array variables of mixed
                shapes,
             -- Open-ended system for adding built-in solution
                methods.
```

Figure 2.1

Key Novel Capabilities of MATHMODEL

MATHMODEL specifications employ the following capabilities:

Unrestricted form of assertions: Assertions can have the form

<expression> >(=) = <(=) <expression>

The user can express identities or constraints as relations between two expressions which naturally represent concepts of the environment (for example, an expression of adding the income variables of a government, plus deficit, is equal to an expression of adding of expense variables). Variables defined or constrained by an assertion need not be typed explicitly by the user, but the type can be determined automatically based on analysis (see subsection 2.2.2). Furthermore, for example, a change to a model which redefines the endogenous and exogenous variables, would not require rewriting the assertions. Simplicity is a key to understanding. The assertions are stated in terms of the application, and they are familar to the user. All subsequent automatic manipulations needed for obtaining a solution, are reported to the user as related to the original form.

Arbitrary order of statements: The underlying notion here is that the user may compose assertions in the order that he or she thinks of them. The user need not indicate the organization of the model by stating the assertions in a particular order or form. The automatic analysis discovers which variables are not defined, or which are redundantly overdefined. The user needs then to make indicated additions or corrections. MATHMODEL provides the user with a report on *matching* each unknown variable with an assertion which defines it. This shows also the completeness of the model in having all the variables properly defined. Next, the clusters of assertions that must be solved together are identified. Causality dependencies are reported as well (see subsection 2.2.2). Relieving the user of these organization tasks is an important help.

Mix of array variables of different shapes: Particularly in a large mathematical model, it is very economical to have structured variables -- such as arrays. (Note that a user may visualize all variables as virtual -- namely as having infinite memory space, while actually an optimizer would generate programs which minimize use of memory, see subsection 2.2.4 for explanation). The main advantage of using array variables is that a single equation may define an entire array variable with a large number of elements. The entire model may include scalars or array

variables of different data types, dimensionalities and sizes of dimensions. The solution methods can be employed automatically on interdependent assertions that involve mixed shapes of array variables. The analysis (see subsection 2.2.2) finds the clusters of assertions that need to be solved simultaneously. The assertions are then manipulated into the formats required by the respective solution methods. The different solution methods typically used in solving a large scale mathematical model are integrated automatically into an overall solution.

Generalized databases and reports: A declaration of the schema of a database or a layout of a report are part of the specification. To reference and update another database or produce a different format of the report, only the declarations need to be modified, not the assertions.

2.2.2. Analysis

The analysis is responsible for constructing a complete and consistent mathematical model, partitioning it and mapping it into respective solution methods. The automatically-conceived organization is then reported to the user. These analysis capabilities do not exist in the traditional mathematical modelling systems [Waren 87]. The analysis steps described below are generic and are based on the mapping of the declarations and assertions into solution methods. The analysis steps are open-ended, easy to add-to as new solution approaches are added to MATHMODEL. They constitute the major aspect of the innovative approach to mathematical modelling.

Under the title of *analysis* we group three inter-related activities: *tolerance of omissions*, *checking* and *organization*. By tolerance of omissions, we mean the parts of the model that the user may omit and must be filled-in automatically. Forcing the user to be explicit about all the details needed for performing the computation is tedious and laborious, which is one of the shortcomings of current mathematical modeliing languages. The filling-in of omissions is based on checking and finding the organizational relations between parts of the mathematical model.

Tolerance of omissions: The tolerance of omission of declarations of internal variable structures is probably one of the greatest labor saving features of MATHMODEL. Only the declarations of input and output data are mandatory; the other variable declarations are optional. The analysis determines the needed precision from the declarations of input and output. From this it derives the data types, length, and scale of the internal variables. Because of efficiency considerations, the precision is limited to that of multiple precision floating data types supported by the object language. The dimensionality and sizes of dimensions are determined from the

references to variables in assertions.

In variables of relatively large number of dimensions, the user may omit referencing in equations the subscripts of the left-most dimensions that apply to all variables in an assertion.

Next, if the mathematical model has variables of the same name which appear with the same values in input and output, it is not necessary to have assertions showing their identity. This is important when using databases of ten (or many more) data elements in a record, with only few of the data elements being updated.

Finally, the user may omit selection of the solution method to be employed in solving simultaneous equations and optimization subproblems, and/or omit choosing the respective solution parameters. The mathematical model is not computationally complete without them. If the user omits specifying them, then they are determined automatically based on the analysis of the respective assertions (section 2.2.3).

Checking: The same process that fills in tolerated omissions also checks consistency of variable dimensions, sizes of dimensions and data types. If they are not used consistently, the user is informed of the offending assertions.

The most important category of checking is related to the concept of completeness of the specification of the mathematical model. In the simplest terms -- there must be equations for determining the values of all the unknown variables. All unknown variables must be referenced by the assertions that define them. An optimization assertion may define more than one variable. This analysis is called *matching*. It identifies a consistent set of unknown variables defined properly by respective assertions. The result of matching is reported to the user to help him/her in comprehending the mathematical model. In certain instances, the user may wish to change the matching in order to speed-up the solution or improve precision. Also, if a match is not feasible, then the variables and assertions that cannot be matched are reported.

Organization of the mathematical model: The most fundamental partial ordering of the assertions and variables of a mathematical model is based on analysis of *causality*, i.e. the dependencies of unknown variables on their defining assertions and dependencies of assertions on their independent variables. This is represented in MATHMODEL by a directed graph. Every variable and every assertion are represented in the graph by a respective node. The dependencies

are represented by edges. An unknown variable node is not at the end of an edge from a defining assertion (i.e. not matched with a defining assertion) indicates an incompleteness error. The graph may contain cycles. The ordering by causality applies then only to groups of assertions and variables in *maximal strongly connected components* (MSCC) in the graph. In the construction and analysis of the graph, MATHMODEL finds the input/output or the single assertions that can define variables, and the clusters of assertions and variables in MSCCs that must be solved simultaneously. Some clusters may also nest within another cluster. This ordering and grouping of variables and assertions are reported to the user. The user may optionally select the solution methods to be employed. Otherwise, for each cluster, the assertions and variables in the MSCC are analyzed to determine an appropriate solution method. The analysis determines the linearity or nonlinearity of the assertions, their formats, whether they define or constrain variables, and whether they involve boolean or regular algebra.

At the end of the analysis, the mathematical model has been fully checked; all discovered omissions have been filled-in automatically or through interaction with the user. All the assertions and variables have been partitioned into components for which a solution method has been selected. At this point, the mathematical model is ready to be evaluated.

2.2.3. Symbolic Manipulation

Traditionally symbolic manipulation has been used to simplify a mathematical model and improve its understanding. It can transform the equations into a form that gives an explicit definition of an unknown variable. In a large model, an explicit definition may not be found by the current methods of symbolic manipulation. Even if found, it would typically be lengthy. Instead, the approach in MATHMODEL is to facilitate the understanding of the model by providing the user with a solution, i.e. values of the variables. Thus, symbolic manipulation is necessary to transform a set of inter-related assertions into a form that is required in the selected numerical solution method. The transformed assertions are of interest to the user for several reasons. First, a user familiar with the solution method may get better understanding of the model by being shown how the assertions have been grouped and transformed into a familiar solution method format. Second, a selected numerical solution method may not be able to produce a solution. The reasons for failure (e.g. non-convergence, etc.) are meaningful to the user in the context of the transformed assertions. The user may thus gain an insight of the reasons of failing to find the solution from examining the transformed assertions. This understanding could lead to

making appropriate changes in the specification (including changes in parameters of the solution method) so that a solution may be found, or so that a solution may be found faster, with better precision, etc. Diverse numerical solution methods require different formats of assertions. Therefore specialized symbolic manipulations must be incorporated for each solution approach. A symbolic manipulation procedure must be added to the mathematical modelling system for each solution method added in the future.

2.2.4. Automatic Programming

Avoiding the need to compose and debug procedural programs: Traditionally, for a given environment, an expert (analyst) composes a mathematical model. In many of the current mathematical modelling systems, programs must be written to implement parts of the model. This procedure is schematically shown as in figure 2.2. Of course, if the problem is very complicated, there may be several levels of analysts, and several levels of programmers. A senior analyst performs the global analysis and several junior analysts refine his/her work. The same situation applies to programmers.

The procedure is different for MATHMODEL. After an analyst completes his/her work, the system is invoked to generate a program. This is shown in figure 2.3. The feedback from program output to the analyst may repeat many times for large and complex system development.

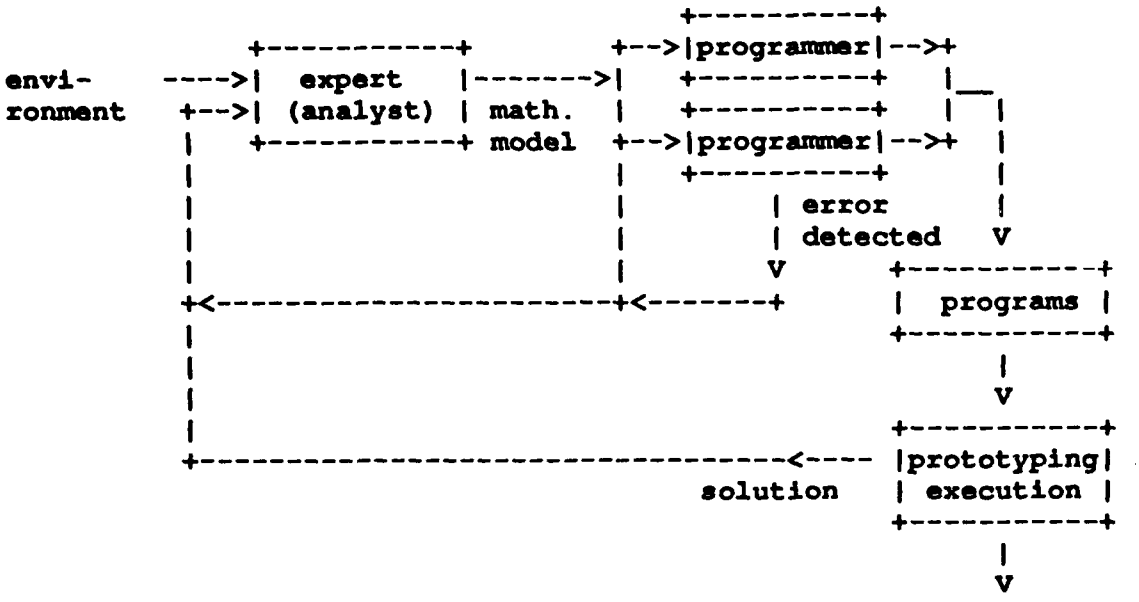

Figure 2.2

Conventional Approach to Computing a Mathematical Model

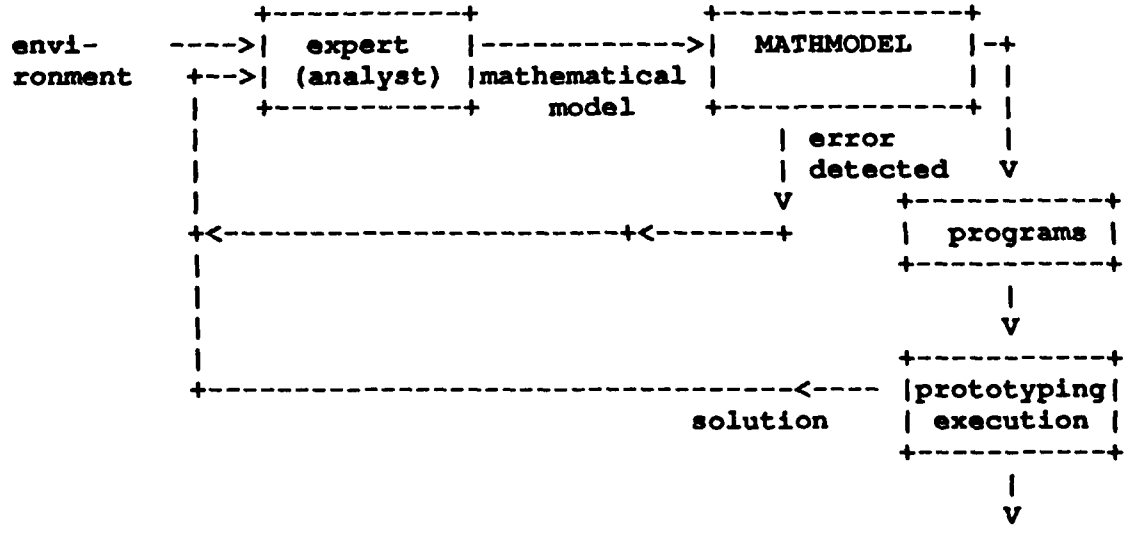

Figure 2.3

New Approach to Computing in MATHMODEL

Generating efficient programs: Efficiency of computation of a solution is critical in large scale mathematical modelling. The inefficiency of many mathematical modelling systems has frequently forced users to use conventional programming approaches. Even though the cost of computation has been decreasing greatly, the issue of efficiency remains paramount due to the increasing size of mathematical models and increasing frequency of their use. Efficiency of computation is particularly critical in mathematical models used for real-time decisions.

The general purpose nature required of the mathematical modelling language makes use of pre-programmed solution methods very difficult. Recent advances in automatic generation of optimized programs are used in MATHMODEL to provide highly efficient customized programs for computation of mathematical models. The notion is to generate programs in a conventional procedural language. The generated programs may be reused, without regenerating them, to repeat evaluation of a model for different exogenous or control variables. In generating programs, MATHMODEL [Prywes 79] systematically examines every variable to minimize use of memory space (maximize sharing of storage locations) and every iteration and control block to minimize control statements and eliminate unnecessary copying. A more complete description of the optimization can be found in [Lu 81].

Testing: The proof of satisfaction of a mathematical model is in its testing; namely, in providing a solution that the user accepts as realistic and useful. The analysis described above checks only some necessary conditions. The testing of the mathematical model with real-life data verifies that it meets the user's intentions and that it is useful for the purpose for which it was developed.

Development of a mathematical model is a trial, error and refinement process in which the mathematical modelling system and the user must interact. For this reason it is necessary that the generated program produce reports that inform the user of the reasons for failing to reach a solution. To interact with the system, the user needs not understand the generated program. The user must, however, comprehend the numerical approach used, the assertions and variables that are involved and how to overcome the encountered problems. To correct the reported problem the user may wish to change the selection of the solution methods, the initial values of variables, the convergence conditions, etc. The testing may also reveal that the mathematical model is redundant and incomplete.

2.2.5. Numerical Analysis

Built-in numerical solution methods are mandatory for an effective mathematical modelling system. They must apply to diverse kinds of assertions -- linear and nonlinear, simultaneous equations and optimizations and different convergence requirements. While pre-programmed highly efficient procedures of solution methods can be used in some cases, the solution of simultaneous equations and optimizations with mixed shape array variables mandate generating customized programs for these cases. Namely, it is not sufficient to write a pre-programmed solution method; it is also necessary to be able to use the method in non-standard combinations of assertions and variable declarations. Both types of numerical solution methods -- pre-programmed and custom generated -- are used in MATHMODEL.

Six solution methods have been incorporated in the MATHMODEL. They are for:

```
Simultaneous Equations -- Gauss Elimination (linear)
                       -- Gauss-Seidel (nonlinear)
                       -- Jacobi (nonlinear)
                       -- Search (nonlinear)

Optimization           -- Simplex (linear)
                       -- Search (nonlinear)
```

MATHMODEL is open-ended to add solution methods. There is much similarity among some methods while others differ greatly. It is necessary to be able to accomodate a variety of new methods. For each new method added it is necessary to be able to easily add capabilities in three categories:

1. Checking that the selected new solution method can be applied to a respective subset of assertions and variables in the specification of a mathematical model.

2. Symbolic manipulation of respective assertions and variables to the form required by the method.

3. Automatically generating the programs that employ the new method.

3. TASK 1: ENHANCEMENTS OF MATHMODEL

3.1. Merger of MATHMODEL With CCCC's MODEL

This task called for integrating the MATHMODEL system developed in research by Dr. X. Ge at the University of Pennsylvania with CCCC's MODEL system. The merger of these two systems was intended primarily to lend to the research version, with its implied unreliability and incomplete portions, the robustness and infrastructure of a commercial version. The merged system has become a better candidate for eventual offering commercially. In fact, the CCCC version of MODEL has a number of additional features which are mandatory for mathematical modelling. Thus, the merger enhanced MATHMODEL with these features, as well. These enhancements are not all of pure technical internal nature. They also affect in a major way the inherent use of MATHMODEL for mathematical modelling. They consist of the following:

- New operations - of matrix and relational algebras.
- Use of relational databases.
- Use of pictorial specification of reports and displays
- Use of a database of specifications for reuse of commonly used data declarations and assertions.
- Generation of test data for testing the generated programs.

In addition, the following enhancements effect the environment in which MATHMODEL is used:

- Use of IBM's computers in addition to Digital's computers.
- Generating programs in C and Ada in addition to PL/1.
- Availability of multitasking to accelerate solution of a model through multiprocessing.

The synthesis of the above capabilities in MATHMODEL involves a large multi-phase system comprised of 142 modules and 60,000 program lines. The remainder of this section describes these phases, which also provide a medium level view of how the MATHMODEL system offers the capabilities enumerated in Section 2.

MATHMODEL is divided into *user implementation* and *execution* phases. These are described in respective subsections.

3.2. User Related Phases of MATHMODEL

The user related part begins with the syntax analysis phase, assuring that the syntax is correct before proceeding with the other phases. The user specification is also transformed into an internal form. This is followed by a *matching* phase where for each equation is found a respective variable which is defined by that equation. The matching also provides a basis in later phases for aggregating all assertions into the smallest blocks which are scheduled most efficiently in the generated program. Next, all information is accumulated in a graph. Using the graph, MATHMODEL proceeds with a phase that checks for ambiguity, completeness, and consistency, resolves the contradictions and, in some cases where it seems appropriate, completes details omitted by the user. At this point, the specification has been thoroughly analyzed. This part consists of 6 phases. The input and output of each phase, their order, and the types of reports or error messages produced are shown in figure 3.1.

The user related phases are as follows.

Phase 1: Syntax Analysis:

The syntax analysis is the first phase of MATHMODEL. The input to this phase is the user's specification. Syntax errors in the specification are detected and reported in this phase. After syntax analysis, the specification is stored for easy retrieval. Besides syntax analysis, this phase also checks the local semantics. The objective is to find as many errors as possible in the early phases.

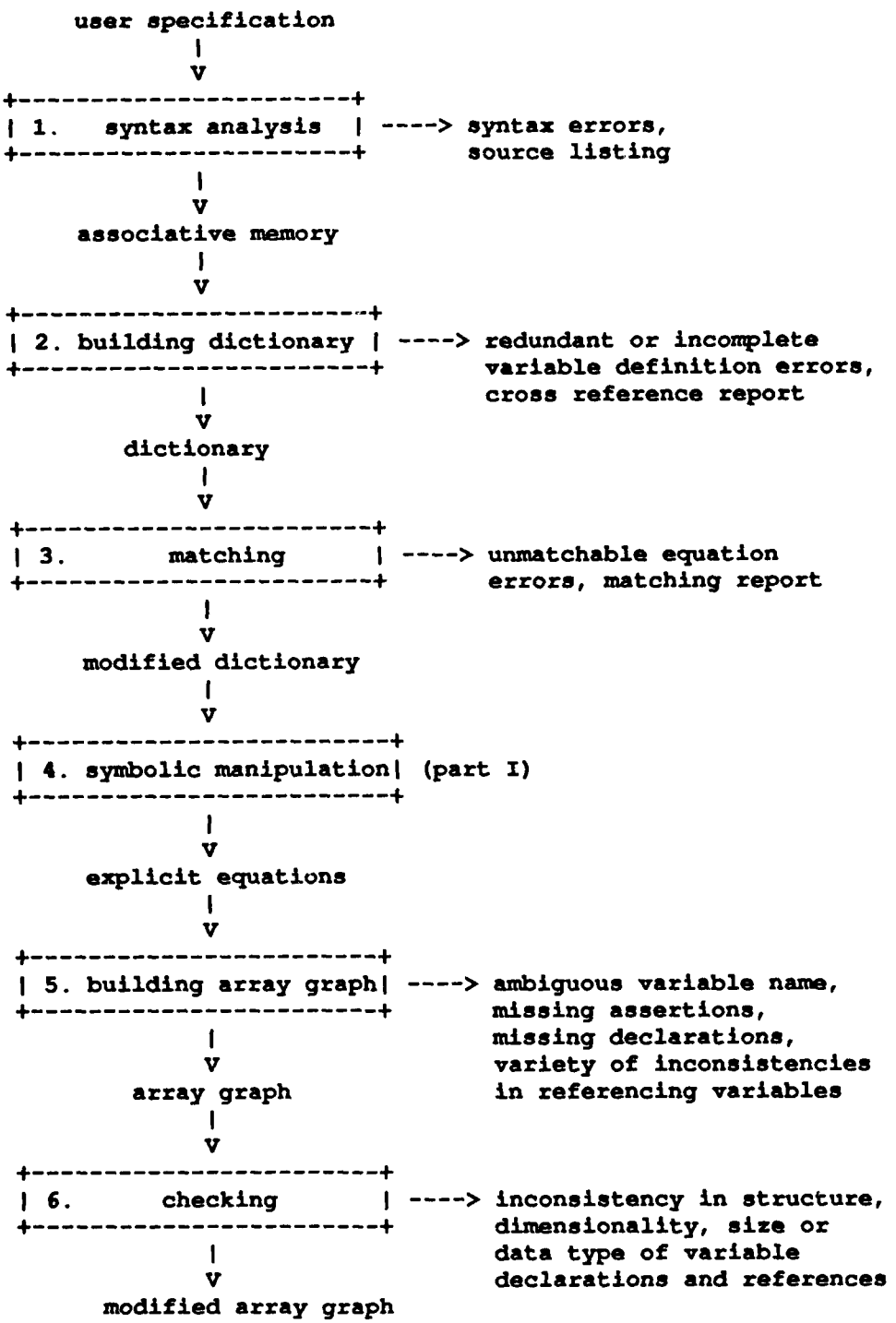

Figure 3.1

User Related Phases of Compiler

Interestingly, the Syntax Analysis Program (SAP) itself is automatically generated by a Syntax Analysis Program Generator (SAPG). The input to SAPG is a formal definition of the MATHMODEL language. This approach makes it very easy to change the language definition, or test a new component of the language.

Phase 2: Building A Dictionary:

The dictionary is a table of contents of total information in the mathematical model. In this phase, a shell is built for the dictionary which is completed in later phases. Using a dictionary greatly reduces later storing and searching.

The dictionary consists of entries for every assertion, every variable, and every subscript. If a name is ambiguous, such as when a variable is declared more than once, it is detected and an error is flagged, if the ambiguity cannot be resolved logically. Other errors in the user specification are detected, such as a control variable with an undefined suffix. Each entry of the dictionary contains many attributes, and these attributes hold the necessary information. Some attributes are filled in at this phase, while others are completed in later phases of analysis as they become available.

Phase 3: Matching Equations with Variables:

The basic notion is that the assertions must be solved in order to evaluate the dependent variables. Thus there must be sufficient assertions to define one or a small set (e.g. in non-linear equations) of solutions for the unknown variables. Namely, if it is obvious at this phase that there are an infinity of possible solutions, this is flagged as an error, and the user is required to add more assertions, or to rewrite the original assertions. To provide guidance in making the correction, each assertion needs to be associated with the variables that it defines. Thus, identifying undefined variables gives the user a guideline on the need to compose additional assertions. Defining dependent variables is also required to attain a highly efficient computation. The matching provides a basis for determining all the dependencies among variables and assertions in later phases. The matching must be performed in the first analysis phase, because practically all the other analysis phases depend on the results of matching.

MATHMODEL allows a user to use an unrestricted form of equations with arithmetic expressions on both sides of the equal sign. It is therefore not known from such an equation, by itself, which variable is defined by this equation. A global analysis of the equation-variable

relationship is necessary to find the variable defined by each equation. The matching algorithm is such a global analysis.

The input to the matching process is the set of unrestricted form equations. The algorithm is adopted from [Hopcro 73].

Phase 4: Symbolic Manipulation (Part I):

The capabilities of symbolic manipulation can be classified into those that concern an individual equation and those that concern multiple assertions that must be manipulated into the required format for a specific solution method. The first class is performed here, and the second class is performed later, after the solution method has been selected.

This phase transforms unrestricted form equations into an explicit form with the unknown variable on the left hand side and an expression defining the variable on the right hand side. An equation can be transformed into an explicit form if it satisfies the following two conditions:

- the equation is not a member of any system of simultaneous equations,
- the unknown variable term either appears only once or could be collected easily.

Phase 5: Array Graph:

In order to analyze and manipulate a specification, there is a need to represent the user's statements in a convenient and accessible internal form. An *array graph* is such an internal form. An array graph, which is very similar to *petri net* or *data flow graph*, accumulates local information from each individual assertion and data declaration statement. All the global analyses, such as checking for consistency, completeness, and ambiguity and scheduling, are performed on the array graph.

A graph is a perfect medium to grasp the fundamental information of a specification. It uses nodes for representing assertions and variables, and edges for representing the precedence relationship among assertions and variables. However, because of the large number of variables and assertions, a naive straightforward graph for representing each element of a variable and each instance of an equation by a node is practically infeasible. Cleverly, MATHMODEL uses one node to express an array of variables or an array of equations. Similarly, an edge represents relationships among all the respective array elements. The information about a whole array of variables or assertions, such as the dimensions of each node, the range of each dimension, and

subscript expressions of each assertion, are recorded as attributes in each node and edge. This special graph is called an *array graph*.

In this phase, MATHMODEL analyzes each file declaration statement, sets up a data node for each data name, and builds edges among data nodes according to the hierarchical relationships. It also analyzes each assertion, sets up an assertion node for each one, and builds edges between the data node and the assertion node according to the dependency relationships. In addition, MATHMODEL sets up a data node for each control variable and builds the edge between the control variable and the affected variables.

After building the array graph, many errors in the user specification can be recognized from the structure of the array graph. For instance, an undefined variable can be easily identified because this data node does not have an incoming edge from an assertion node. Using the properties of the array graph, this phase is able to find many types of errors in the user specification.

Phase 6: Checking and Propagation

The purpose of checking and propagation is to

- recognize all missing attributes of nodes (which represent the tolerated missing information),
- find the incomplete or missing information in a user specification by propagating information from other assertions or variables,
- flag missing information that cannot be deduced from another source as an incompleteness error, and
- flag conflicting information as an inconsistency error.

The checking is performed on the array graph and all deduced information is stored as attributes of the nodes and edges of the array graph.

The checking consists of *dimension propagation*, *range propagation*, and *data type propagation*.

Dimension propagation: although each variable in the source or target data

has a clear dimension definition. Many interim variables used in the specification may not have such a definition at all. A user may also mistakenly miss subscripts or intentionally omit subscripts in assertions to keep the specification shorter and easier to read. Dimension propagation checks the use of subscripts, provides the omitted subscripts, identifies the misuse of subscripts, and finally provides each variable and each assertion with a consistent dimension definition.

Range propagation: each dimension of each array variable or assertion must have its range defined (or more accurately, have its size defined). This range may be defined from the source or target data declaration statement, control variables, the actual size of the data, or from the range definition of other nodes. A propagation strategy is used to find a unique range definition for every dimension of each variable or assertion. If any range is not defined directly and cannot be propagated from other ranges, or if the range has more than one definition, MATHMODEL flags it as an error. If the range can be propagated from two different sources, the compiler resolves the contradiction based on efficiency considerations.

Data type propagation: each input or output variable has to have a data type supplied in the declaration. The interim variables may not have data types declared at all. When using operations to manipulate these variables, there is a problem of using data types consistently. For instance, it is meaningless to add a character string to a decimal number, or to have a character string assigned to a decimal number. On the other hand, if one character string is equal to a variable which does not have a declared data type, it is reasonable to assume that the latter has a character string data type as well. Data type propagation checks the consistent use of data types in each assertion and expression and defines a data type for each variable which does not have a clearly declared data type definition when it can be propagated from another variable. It also flags errors if it finds an inconsistency.

When MATHMODEL cannot propagate the missing information or resolve the conflicting information, it flags an error. In the error message it gives the reason for the error, the assertions involved in the error, and seeks assistance from the user.

3.3. Implementation Related Part of MATHMODEL

Using the previous results, the implementation related part can proceed. It first partitions all assertions into the smallest blocks (each block is an MSCC) and organize then them in an optimum order in terms of using memory and execution time for the respective generated program. This leads to generating a schedule of computation events in the form of a flowchart. Analyzing the blocks, it is possible to detect additional types of errors in each block which are caused by circularity in the user specification. If no error is detected, a solution method is selected for each individual block. The code generation phase finally transforms the events in the flowchart, one after another, into respective sub-programs. It then merges the sub-programs together, and produces a complete conventional language program. This part consists of 4 phases. The input and output of each phase, their connections, and the report and error messages produced in each phase are shown in figure 3.2.

The implementation related phases are discussed below.

Phase 7: Scheduling:

MATHMODEL allows a user to choose a representation for his problem **in the most natural and convenient way for that problem.** But this usually does not correspond to the most efficient way to perform the computation. The scheduling bridges this gap by taking a user specification in the form of an array graph and producing from it an optimal *flowchart* in terms of minimizing memory and execution time in the generated program.

Using the array graph as input, the scheduling produces a flowchart as output. The flowchart corresponds to the execution order in the conventional language program that will be generated.

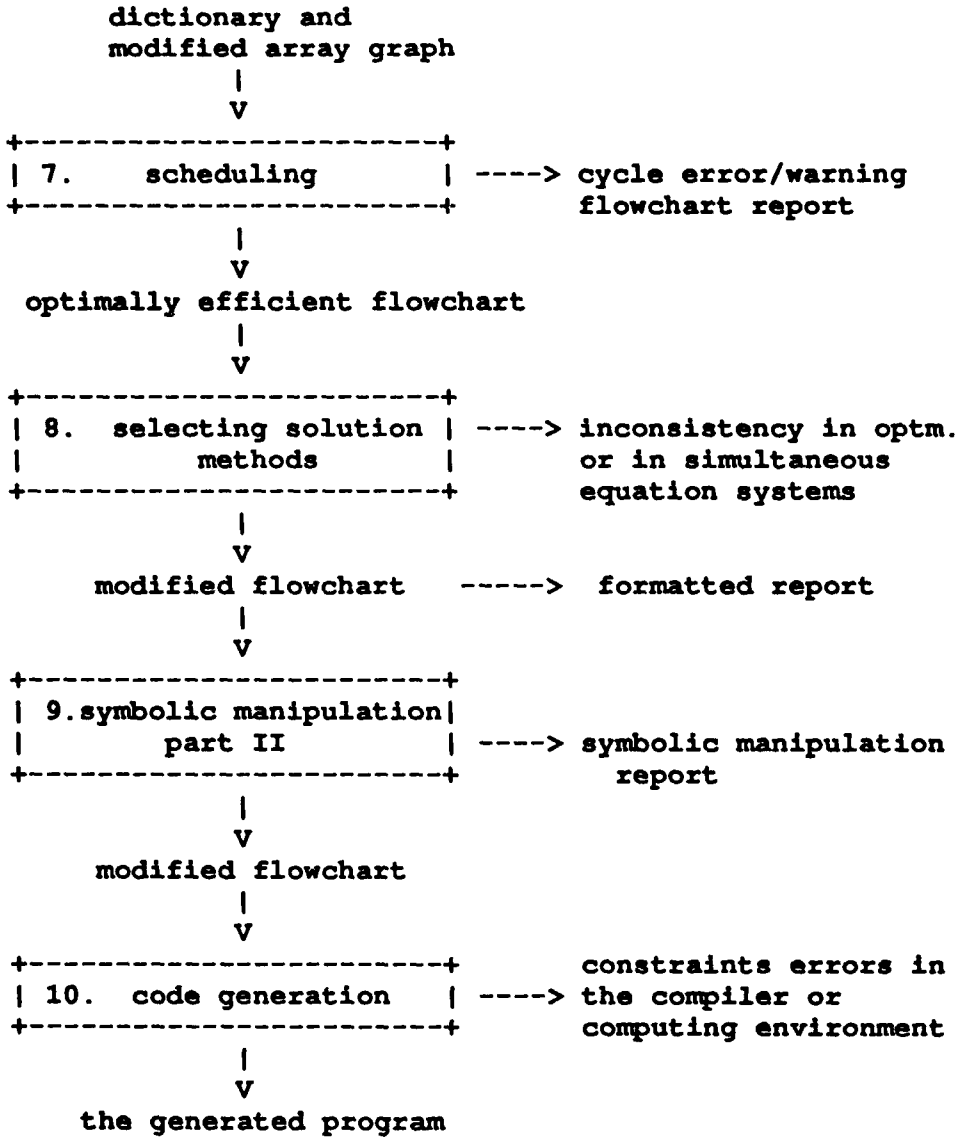

Figure 3.2

Implementation Related Phases of the Compiler

The flowchart is recursively defined as a sequence of linear order *elements* that may be nested. That is, at the highest level, it is a sequence of linear order elements. In turn, each element consists of a sequence of linear order elements, and so on. The element, on the one hand, is an aggregation of statements of the user specification. On the other hand, it represents a computation event which will be translated into a piece of conventional language code in the later phase.

The following are four different kinds of elements, their corresponding statements in the user specification, and their meaning in terms of the computation events.

1. node-element: a node-element is a terminal element. It represents an assertion or a data statement. It may correspond to an assignment, or an I/O operation in the code generation.

2. for-element: a for-element is a structured element which can be recursively redefined as a sequence of other elements. A for-element corresponds to a for-loop in the generated program.

3. simul-element: as with a for-element, a simul-element is a structured element. A set of simultaneous equations or an optimization with its related constraints constitute a simul-element. This element corresponds to a computation event which includes the solution method, the related parameters, and the mathematical formulas used in the solution methods. The simul-element contains the attributes of initial value, iteration number and other parameters for the solution method. If these are given in the user specification, they are added into the simul-element. Otherwise, they will be decided and filled in by the compiler in the next phase.

4. cond-element: a cond-element is a structured element which corresponds to a conditional equation. A cond-element consists of one or two elements which correspond to the 'then' and 'else' parts of a condition.

Two mutually recursive algorithms are used to produce a flowchart from an array graph. The first algorithm finds and schedules all the MSCCs of the array graph. The second algorithm takes each MSCC as a single node and performs a topological sort. Then, for each MSCC the first and second algorithms are called again to decompose the array graph further. This process is used recursively until a terminal node is reached. A straightforward sorting may cause the generated program to be very inefficient. A global optimization technique is used in the topological sort to minimize the use of memory space in the generated program, and consequently the execution time is also minimized.

Phase 8: Selecting Solution Methods:

The scheduling in the last phase puts each system of simultaneous equations or each optimization assertion and related constraints together in the form of a block. Each such block forms a simul-element in the flowchart. This phase analyzes this simul-element in the flowchart, and either applies the user specified solution method or automatically selects a solution method. The method is selected based on distinguishing optimization from simultaneous equation problems and linear from nonlinear problems. The criteria for selection of solution method is shown in figure 3.3.

	linear	non-linear
optimization	Simplex	Search
simultaneous equations	Gauss Elimination	Gauss-Seidel Search

Figure 3.3
Criteria for Selecting Solution Methods

If there is more than one optimization assertion in one block, then the mistake is reported to the user.

Phase 9: Symbolic Manipulation (Part II):

Some solution methods require a specific format for the assertions.

For the Gauss Elimination method, the equations have to be in the form of

```
A X = B
```

Here A is the matrix of coefficients; X is the vector of the unknown variables and B is the vector of right hand side constants.

For linear programming, the format has to be

```
        minimize(C X)
subject to:
        A X <= B
```

Here X is the vector of decision variables; C is the vector of the coefficients for the objective function; B is the matrix of coefficients of the constraints and B is the vector of right hand side constants for the constraints.

For both search methods, an assertion

$$f(X) = g(X)$$

is transformed into

$$F(x) = 0$$

Here X represents all the variables (both dependent and independent) appearing in the assertion.

For Gauss-Seidel and Jacobi methods the user is required to write the equation with the dependent variable on the left hand side. If the user does not write the equation in the correct form, this will be flagged as an error.

A user may add new solution methods. If a new solution method does not need a new format, then no new symbolic manipulation is needed. However, if a new solution method needs a new format, such as the Newton-Raphson method for nonlinear equations, the symbolic manipulation capabilities must also be added.

Phase 10: Code Generation:

After scheduling, a flowchart is produced. For each kind of elements in the flowchart, there corresponds a specific piece of code. This correspondence can be summarized as follows:

- **a node-element:** assertion node: this corresponds to an assignment in the conventional language program. However, since the same memory element may be reused, the corresponding subscripts may have to be deleted in the generated program.
- **file node:** if this is source data, it corresponds to opening the source file statement; if this is target data, it corresponds to closing the file statement. The file node also produces some declaration statements which provide the variables to be used in the executable statements.
- **record node:** if this record is in a source file, it corresponds to reading a record from a source file to a record buffer; if this record is in a target file, it corresponds to writing the record buffer into the target file. The record node also produces some declaration statements which declare the related buffers
- **field node:** the code for a field node depends on whether the parent record node should be packed or unpacked. If it does not need to be packed or unpacked, the field node has no corresponding code. Otherwise, if this field is in a source file, it corresponds to copying the value from the record buffer; if this field is in a target file, it corresponds to copying the value to record buffer. The field node also produces the declaration statement which declares the corresponding field variable.

Each for-element corresponds to a for-loop or a while-loop statement. If the range of the corresponding loop is a constant, or is defined by a size-prefixed control variable, a for-loop is used. If the range is defined by a end-prefixed control variable, a while-loop is used. The

contents inside the loop are from the constituents of the for-element.

Each simul-element corresponds to a linear program, a nonlinear program, or a system of equations. The simul-element contains the solution method, the parameters, and the related assertions. The solution methods allowed were shown in Figure 3.3.

The format of each of the above solution methods is different. They are classified as:

1. Using the assertions to generate the executable code directly; Gauss-Seidel, Jacobi, and both search methods belong to this class,
2. Using the assertions to generate the necessary parameters which are used to call the pre-programmed subroutines; Gauss elimination and simplex method belong to this class.
3. Each condition-element corresponds to an if-then-else structure. The constituents of the if-then-else are from the constituents of the condition element.

The code generation phase simply takes elements from the flowchart one after another, analyzes them, and produces a specific piece of code in a conventional language. The dictionary and the syntax tree of each assertion provide an important information source for this phase.

3.4. Execution Related Phases of MATHMODEL

After compilation, a conventional language program is generated. A conventional language compiler and linker are used to produce an equivalent machine code program. After executing this program, the variables in the target files have been evaluated.

The execution related phases are shown in figure 3.4.

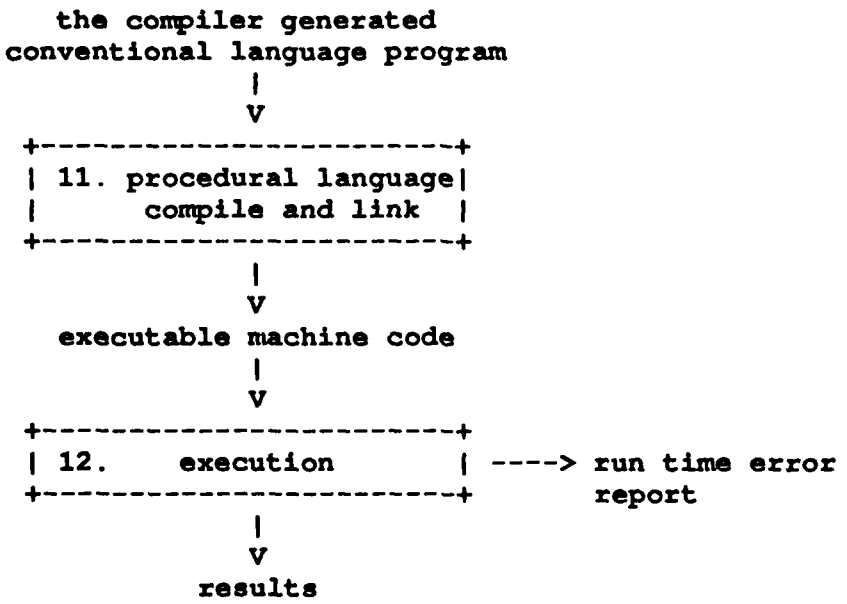

Figure 3.4

Execution related Phases of MATHMODEL

The following two cases may occur:

1. The variables cannot be evaluated, or

2. Although evaluated, the user does not like the results and wants to make changes to the mathematical model.

These usually happen when:

- The mathematical model is wrong,
- The source files have mistakes,
- The parameters in the solution methods are not properly chosen.

In order to help the user to determine exactly what and where the problem is, the programs produce runtime error messages. The runtime error messages include:

The block name, which indicates the place where the problem happened,

The position of the assertion

A hint to reset parameters.

4. TASK 2: DEMONSTRATING THE USE OF MATHMODEL

4.1. Objectives of the Demonstrations

The objective of Task 2 has been to produce evidence needed to convince potential users of the advantages of MATHMODEL. Three approaches to meeting this objective are discussed in the subsequent subsections, as follows:

1. *Explaining the underlying methodology of using* MATHMODEL. Mathematical modelling has a discipline that the developer must follow, starting from the establishing of requirements to providing answers to the questions asked of a model. Explaining how MATHMODEL fits into this discipline is an important step to meeting the above objective. This is described in Section 4.2.

2. Use of small examples. The example can be used for training purposes, to illustrate the language and the operation of MATHMODEL. Three small examples are given in Section 4.3. They are also intended to present and illustrate MATHMODEL to the reader of this report.

3. Use of Large Examples. MATHMODEL may be used in large mathematical modelling projects to develop one or several components - typically a procedure or a task. This goes beyond the scope of this report. A summary of another project at CCCC where such a development took place is provided in Section 4.4.

4.2. Use of MATHMODEL

This section explains how to use the MATHMODEL system. Figure 4.1 explains seven steps in using MATHMODEL. This is followed by descriptions of each of these steps.

Steps in Using the System:

1. A user formulates a mathematical modelling problem. He provides mathematical formulas (equations, optimizations, inequalities, and equalities) to describe a physical or social process and some data from observations or experiments in a database. He needs to describe values of variables which represent the model.

2. The user represents the model as a specification, which consists mainly of the

mathematical formulas themselves.

3. The user submits this specification to MATHMODEL. It checks the input specification for completeness, consistency, and ambiguity, partitions the user specification into the smallest blocks, selects a solution method for each block, and maps the user's assertions into each solution method. If any of these fails, error messages are produced. Otherwise, a conventional language program is generated.

4. MATHMODEL produces documentation. A user can select various reports, including: a listing of the specification, an equation-variable match report, a cross-reference report, a subscript-range report, a flowchart report, a listing of the generated program, and an error and warning report. Analyzing these reports, a user may wish to go back to step 2 and modify the specification. If everything is acceptable, he proceeds to the next step.

5. After having successfully generated a conventional language program, the user can submit it to the conventional language compiler and load it in preparation for execution. Before running the program, he must have his source data files.

6. After running the program and examining the results, the job is complete if the user is satisfied with the results.

7. If the user is not satisfied with the results, he can alter the specification and return to step 2.

MATHMODEL is designed to accept mathematical formulas directly. Therefore, using the mathematical formulas, the user simply adds a header and data declarations to form a complete specification. The header gives the module name, and source and target file names. The data declarations give the structure of the source and target files.

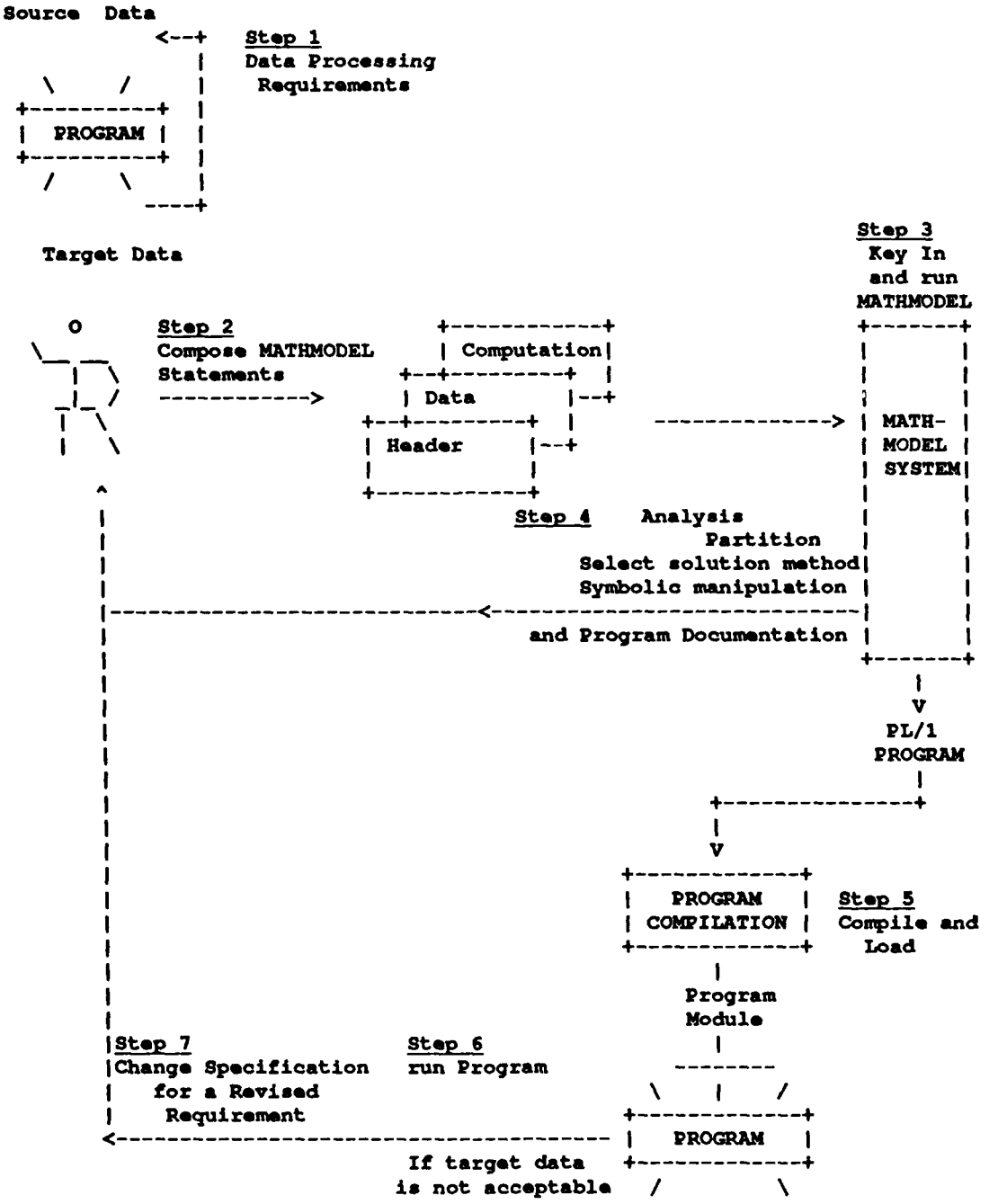

Figure 4.1

The Overall Procedure for Using MATHMODEL

It is the user's responsibility to modify the mathematical model, correct the data, or change the parameters. In the generated program, MATHMODEL includes code to produce a report that provides the user with essential information on the problems encountered during execution and how to overcome them.

There are 6 solution methods built into the system. The possible execution error phenomena and its possible reasons for built-in solution methods are listed in figure 4.2.

Numerical Methods	Phenomena	Reasons
Gauss Elimination	elimination can not continue	coefficient matrix singular
Simplex	objective goes to infinity	constraints are not properly given
Jacobi	not convergent	iterations, relative error are not properly given, or the problem does not converge
Gauss-Seidel	not convergent	iterations, relative error are not properly given, or the problem does not converge
search (nonlinear equations)	not convergent	iterations, relative error are not properly given, or the problem does not converge
search (nonlinear programming)	not convergent	iterations, relative error are not properly given, or the problem does not converge

Figure 4.2
Runtime Error Messages and Their Reasons

If any of these errors occur, the generated program prints the error message to give the position and reason for the error. According to the error message, the user might need to correct the mathematical model, or change the parameters in the solution method, then recompile the specification again. This corresponds to step 7 in figure 4.1.

4.3. Illustration of MATHMODEL Through Small Examples

Three examples related to an electrical circuit are used in this section to explain the power and the concept of MATHMODEL. Complete MATHMODEL reports of the first example and partial MATHMODEL reports of the other examples are also given below.

The circuit is shown in figure 4.3. It is designed for use on a V1-volt source of electromotive force in charging V2-volt and V3-volt batteries connected in parallel. The symbols V1, V2, V3, R1, R3, I1, I2, I3 represent the values as shown on figure 4.3.

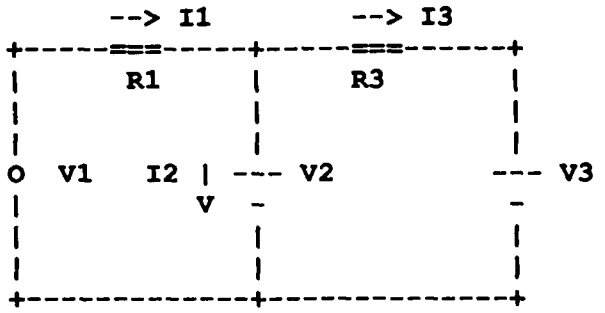

Figure 4.3
V1 Source Charges V2 and V3 Batteries

Create a mathematical model of the circuit to answer the following questions:

- **Question 1:** If the currents I1, I2 are given, find the values of R1 and R2.

- **Question 2:** If the purpose is to maximize the output power,

  `W=V2*I2+V3*I3 ,`

  find the values of I1, I2 and I3. From them, find the values of R1 and R3.

- **Question 3:** Suppose that

  ```
  power=k1*W=k1*(V2*I2+V3*I3), and
  cost=k2*V1*I1+k3*(V1*I1-W)
  ```

  Here the cost has two terms. The first term is the power consumed, the second term is the cost for cooling two resistors R1 and R3; Maximize the cost/power. Then find the values of R1 and R3.

Question 1: Composition of the mathematical model:

According to the Kirchoff laws, we have

```
I1*R1=V1-V2;
I1*R1+I3*R3=V1-V3;
I1=I2+I3;
```

These three equations contain all the information to answer question 1. They are submitted to MATHMODEL to solve R1 and R3 from given V1, V2, V3, I2, I3.

Figure 4.4 (a): The user specification for Question 1

```
module: circuit;
source: param;
target: design;

1 param is file,
  2 inr is rec,
    3 (v1,v2,v3,i2,i3) are field(pic 'zzz9v.99');

1 design is file,
  2 outr is rec,
    3 (r1,r3,i1,w) are field(pic 'zzz9v.99');

i1*r1=v1-v2;
i1*r1+i3*r3=v1-v3;
i1=i2+i3;
w=v2*i2+v3*i3;
```

Figure 4.4 (b): Reports

```
                                        ***** SOURCE LISTING *****

MODEL PROCESSOR: VERSION 7.6:R4 WITH BLOCK STRUCTURE ON VAX 11/750    OCTOBER 27, 1987      09:02:55.69

File Name: Q1.INP

                          ************************************************************************
                          *                                                                      *
                          *                    CIRCUIT MODULE SPECIFICATION                      *
                          *                                                                      *
                          ************************************************************************

   1      MODULE: CIRCUIT;
   2      SOURCE: PARAM;
   3      TARGET: DESIGN;


                          ************************************************************************
                          *                                                                      *
                          *                         FILE DESCRIPTIONS:                           *
                          *                                                                      *
                          ************************************************************************


                          ************************************************************************
                          *                                                                      *
                          *                    DESCRIPTION OF PARAM FILE                         *
                          *                                                                      *
                          ************************************************************************

   4      1 PARAM IS FILE,
   4        2 INR IS REC,
   4          3 (V1,V2,V3,I2,I3) ARE FIELD(PIC 'zzz9v.99');


                          ************************************************************************
                          *                                                                      *
                          *                    DESCRIPTION OF DESIGN FILE                        *
                          *                                                                      *
                          ************************************************************************

   5      1 DESIGN IS FILE,
   5        2 OUTR IS REC,
   5          3 (R1,R3,I1,W) ARE FIELD(PIC 'zzz9v.99');

   6      I1*R1=V1-V2;
   7      I1*R1+I3*R3=V1-V3;
   8      I1=I2+I3;

   9      W=V2*I2+V3*I3;
```

Figure 4.4 (b) Continued

```
                                        *****  SOURCE LISTING  *****

**********SYSTEM GENERATED STATEMENT(S)**********

                                    *****  ASSERTION VARIABLE MATCH REPORT  *****

        NAME            DESCRIPTION                         RELATED VARIABLES
        ----            -----------                         -----------------

        AAS86           VARIABLE DEFINED EXPLICITLY         DESIGN.R1
        AAS87           VARIABLE DEFINED EXPLICITLY         DESIGN.R3
        AAS88           VARIABLE DEFINED EXPLICITLY         DESIGN.I1
        AAS89           VARIABLE DEFINED EXPLICITLY         DESIGN.W
                                *****  CROSS REFERENCE AND ATTRIBUTES REPORT  *****

                          WHERE
  NAME                    DECLARED        ATTRIBUTES                          STATEMENT NUMBER REFERENCE
  ----                    --------        ----------                          ----------------------------

CIRCUIT                         1  MODULE NAME
DESIGN                          5  FILE,TARGET,UNSORTED                     3
I1                              5  FIELD, PICTURE'zzz9v.99' IN FILE DESIGN  5,   6,   7,   8

I2                              4  FIELD, PICTURE'zzz9v.99' IN FILE PARAM   4,   8,   9

I3                              4  FIELD, PICTURE'zzz9v.99' IN FILE PARAM   4,   7,   8,   9

INR                             4  RECORD,( 5 SUB-MEMBERS), IN FILE PARAM   4

OUTR                            5  RECORD,( 4 SUB-MEMBERS), IN FILE DESIGN  5

PARAM                           4  FILE,SOURCE,UNSORTED                     2
R1                              5  FIELD, PICTURE'zzz9v.99' IN FILE DESIGN  5,   6,   7

R3                              5  FIELD, PICTURE'zzz9v.99' IN FILE DESIGN  5,   7

V1                              4  FIELD, PICTURE'zzz9v.99' IN FILE PARAM   4,   6,   7

V2                              4  FIELD, PICTURE'zzz9v.99' IN FILE PARAM   4,   6,   9

V3                              4  FIELD, PICTURE'zzz9v.99' IN FILE PARAM   4,   7,   9

W                               5  FIELD, PICTURE'zzz9v.99' IN FILE DESIGN  5,   9

                                        *****  FLOWCHART REPORT  *****

                          NEST
NAME                      LVL: DESCRIPTION                    EVENT
----                      ---- -----------                    -----

CIRCUIT                        MODULE NAME                    PROCEDURE HEADING
PARAM                          FILE                           OPEN FILE
INR                            RECORD IN FILE PARAM           READ RECORD
V1                             FIELD IN RECORD INR
V2                             FIELD IN RECORD INR
V3                             FIELD IN RECORD INR
I2                             FIELD IN RECORD INR
I3                             FIELD IN RECORD INR
AAS89                          EQUATION
W                              FIELD IN RECORD OUTR           TARGET OF EQUATION: AAS89
AAS88                          EQUATION
I1                             FIELD IN RECORD OUTR           TARGET OF EQUATION: AAS88
AAS86                          EQUATION
R1                             FIELD IN RECORD OUTR           TARGET OF EQUATION: AAS86
AAS87                          EQUATION
R3                             FIELD IN RECORD OUTR           TARGET OF EQUATION: AAS87
OUTR                           RECORD IN FILE DESIGN          WRITE RECORD
DESIGN                         FILE                           CLOSE FILE
                               END
```

Figure 4.4 (b) Continued

```
                                        *****  RANGE TABLE  *****

no dimension specifications

                                        *****  FORMATTED REPORT  *****

                        ************************************************************************
                        *                                                                      *
                        *                    CIRCUIT MODULE SPECIFICATION                      *
                        *                                                                      *
                        ************************************************************************

    MODULE: CIRCUIT;
    SOURCE: PARAM;
    TARGET: DESIGN;

                        ************************************************************************
                        *                                                                      *
                        *                          DATA DESCRIPTION:                           *
                        *                                                                      *
                        ************************************************************************


                        ************************************************************************
                        *                                                                      *
                        *                      DESCRIPTION OF PARAM                            *
                        *                                                                      *
                        ************************************************************************

    1 PARAM IS FILE,
      STORAGE NAME IS NSTGNM1,
      ORG IS SAM,
      2 INR IS RECORD ,
        3 V1 IS FIELD (PIC 'zzz9v.99'),
        3 V2 IS FIELD (PIC 'zzz9v.99'),
        3 V3 IS FIELD (PIC 'zzz9v.99'),
        3 I2 IS FIELD (PIC 'zzz9v.99'),
        3 I3 IS FIELD (PIC 'zzz9v.99');

                        ************************************************************************
                        *                                                                      *
                        *                      DESCRIPTION OF DESIGN                           *
                        *                                                                      *
                        ************************************************************************

    1 DESIGN IS FILE,
      STORAGE NAME IS NSTGNM2,
      ORG IS SAM,
      2 OUTR IS RECORD ,
        3 R1 IS FIELD (PIC 'zzz9v.99'),
        3 R3 IS FIELD (PIC 'zzz9v.99'),
        3 I1 IS FIELD (PIC 'zzz9v.99'),
        3 W IS FIELD (PIC 'zzz9v.99');

           /* ASSERTION(S) FOR FILE(DESIGN) */

     /*8*/
    DESIGN.I1 =@PARAM.I2 +PARAM.I3 ;

     /*6*/
    DESIGN.R1 =(PARAM.V1 -PARAM.V2 )/DESIGN.I1 ;

                                        *****  FORMATTED REPORT  *****

     /*7*/
    DESIGN.R3 =(PARAM.V1 -PARAM.V3 -DESIGN.I1 *DESIGN.R1 )/PARAM.I3 ;

     /*9*/
    DESIGN.W =@PARAM.V2 *PARAM.I2 +PARAM.V3 *PARAM.I3 ;

                        ************************************************************************
                        *                                                                      *
                        *                      END OF FORMATTED REPORT                         *
                        *                                                                      *
                        ************************************************************************
```

Figure 4.4 (c)

```
                /****************************************/
                /*            PL/I PROGRAM              */
                /****************************************/
CIRCUIT: PROCEDURE OPTIONS(MAIN);
DCL @NULSTR CHAR(1);
DCL PARAMS RECORD SEQL INPUT;
DCL @FSTPARAMS BIT(1) INIT('1'B);
DCL ENDFILE@PARAMS BIT(1) INIT('0'B);
DCL @VB42 CHAR(35) VARYING INIT('');
DCL @FX42 FIXED BIN(31,0);
DCL @RV43 CHAR(35) BASED(ADDR(PARAM)) ;
DCL @RV37 CHAR(28) BASED(ADDR(DESIGN)) ;
DCL DESIGNT RECORD SEQL OUTPUT ENV(MAXIMUM_RECORD_SIZE (28));
DCL @FSTDESIGNT BIT(1) INIT('1'B);
OPEN FILE(DESIGNT) OUTPUT;
DCL @ERROR_BUF CHAR(270) VAR;
DCL ERRORF FILE RECORD OUTPUT;
DCL ERRORF_BIT BIT(1) STATIC INIT('1'B);
DCL @ERROR FIXED BIN(15,0) INIT(0);
DCL @NOT_DONE(20) BIT(1);
DCL @TMP_VAL FLOAT BIN;
DCL (@RD_LP@,@R_L) LABEL;
DECLARE
   1 DESIGN,
    2 OUTR,
     3 R1 PIC'zzz9v.99',
     3 R3 PIC'zzz9v.99',
     3 I1 PIC'zzz9v.99',
     3 W PIC'zzz9v.99';
DECLARE
   1 PARAM,
    2 INR,
     3 V1 PIC'zzz9v.99',
     3 V2 PIC'zzz9v.99',
     3 V3 PIC'zzz9v.99',
     3 I2 PIC'zzz9v.99',
     3 I3 PIC'zzz9v.99';
DCL (TRUE,SELECTED) BIT(1) INIT('1'B);
DCL (FALSE,NOT_SELE,NOT_SELECTED) BIT(1) INIT('0'B);
ON ENDFILE(PARAMS) BEGIN;
ENDFILE@PARAMS='1'B;
 @VB42=COPY(' ',35);
 END;
ON UNDEFINEDFILE(ERRORF)  ERRORF_BIT='0'B;
DECLARE PLI@_CNVERR GLOBALREF VALUE FIXED BIN(31);
DECLARE RMS@_RLK GLOBALREF VALUE FIXED BIN(31);
ON ERROR BEGIN;
IF ONCODE()=RMS@_RLK THEN GOTO @RD_LP@;
IF @ERROR=0 THEN CALL RESIGNAL();
IF ONCODE()=PLI@_CNVERR THEN DO;
  @ERROR=1;
  IF ERRORF_BIT & @ERROR>0 THEN WRITE FILE(ERRORF) FROM (@ERROR_BUF);
 END;
  ELSE CALL RESIGNAL();
END;
OPEN FILE(PARAMS) INPUT SEQL RECORD;
@ERROR=1;
READ FILE(PARAMS) INTO (PARAM);
  IF @ERROR=0 THEN @ERROR=1;
@ERROR_BUF=@RV43;

                /****************************************/
                /*            PL/I PROGRAM              */
                /****************************************/
/* 9 */DESIGN.W=PARAM.V2*PARAM.I2+PARAM.V3*PARAM.I3;
/* 8 */DESIGN.I1=PARAM.I2+PARAM.I3;
/* 6 */DESIGN.R1=(PARAM.V1-PARAM.V2)/DESIGN.I1;
/* 7 */DESIGN.R3=((PARAM.V1-PARAM.V3)-DESIGN.I1*DESIGN.R1)/PARAM.I3;
WRITE FILE(DESIGNT) FROM (@RV37);
CLOSE FILE(DESIGNT);
RETURN;
END CIRCUIT;
```

Question 2:

Question 2 can be solved by linear programming. It can be expressed as an optimization function with some additional constraints (the constraints restrict the solution domain).

```
w = maximize( V2*I2+V3*I3) dec_var  I2, I3;
I2<=6;
I3<=4;
I2,I3>=0;
```

From this linear programming, the values of I1 and I2 can be solve. Substitute these values into the two equations in the solution for question 1 above, we can get a set of complete assertions for question 2.

Figure 4.5 (a) The user specification for Question 2

```
module: circuit;
source: param;
target: design;

1 param is file,
  2 inr is rec,
    3 (v1,v2,v3) are field(pic 'zzz9v.99');

1 design is file,
  2 outr is rec,
    3 (r1,r3,i1,i2,i3,w) are field(pic 'zzz9v.99');

i1*r1=v1-v2;
i1*r1+i3*r3=v1-v3;
i1=i2+i3;

w=maximize(v2*i2+v3*i3) variable i2,i3;
i2<=4;
i3<=4;
i2+i3<=6;
i2,i3>=0;
```

Figure 4.5 (b) The flowchart report for Question 2

```
                                  ***** FLOWCHART REPORT *****

                          NEST
NAME                      LVL: DESCRIPTION                          EVENT
----                      ---- -----------                          -----

CIRCUIT                        MODULE NAME                          PROCEDURE HEADING
PARAM                          FILE                                 OPEN FILE
INR                            RECORD IN FILE PARAM                 READ RECORD
V1                             FIELD IN RECORD INR
V2                             FIELD IN RECORD INR
V3                             FIELD IN RECORD INR
                             1 METHOD: SIMPLEX ($MAINBLOCK          MATHEMATICAL PROGRAMMING BLOCK
I2                             FIELD IN RECORD OUTR
I3                             FIELD IN RECORD OUTR
AAS89                          EQUATION
W                              FIELD IN RECORD OUTR                 TARGET OF EQUATION: AAS89
AAS810                         INEQUALITY CONSTRAINT
$D$1                           FIELD
AAS811                         INEQUALITY CONSTRAINT
$D$2                           FIELD
AAS812                         INEQUALITY CONSTRAINT
$D$3                           FIELD
AAS813                         INEQUALITY CONSTRAINT
$D$4                           FIELD
AAS813AB                       INEQUALITY CONSTRAINT
$D$5                           FIELD
                             1 END METHOD: SIMPLEX ($MAINBLOCK      MATHEMATICAL PROGRAMMING BLOCK
AAS88                          EQUATION
I1                             FIELD IN RECORD OUTR                 TARGET OF EQUATION: AAS88
AAS86                          EQUATION
R1                             FIELD IN RECORD OUTR                 TARGET OF EQUATION: AAS87
AAS87                          EQUATION
R3                             FIELD IN RECORD OUTR                 TARGET OF EQUATION:   AAS87
OUTR                           RECORD IN FILE DESIGN                WRITE RECORD
DESIGN                         FILE                                 CLOSE FILE
                               END
```

Figure 4.5 (c): The generated program for Question 2

```
CIRCUIT: PROCEDURE OPTIONS(MAIN);
DCL $MALSTR CHAR(1);
DCL PARAMS RECORD SEQL INPUT;
DCL $FSTPARAMS BIT(1) INIT('1'B);
DCL ENDFILE$PARAMS BIT(1) INIT('0'B);
DCL $FB49 CHAR(21) VARYING INIT('');
DCL $FX49 FIXED BIN(31,0);
DCL $RV50 CHAR(21) BASED(ADDR(PARAM)) ;
dcl ($no_of_ass,$no_of_var) fixed bin;
dcl simplex entry((*,*) float bin(53), fixed bin,fixed bin, (*)fixed bin,fixed
bin,(*)float bin(53));
DCL $RV42 CHAR(42) BASED(ADDR(DESIGN)) ;
DCL DESIGNT RECORD SEQL OUTPUT ENV(MAXIMUM_RECORD_SIZE (42));
DCL $FSTDESIGNT BIT(1) INIT('1'B);
OPEN FILE(DESIGNT) OUTPUT;
DCL $ERROR_BUF CHAR(270) VAR;
DCL ERRORF FILE RECORD OUTPUT;
DCL ERRORF_BIT BIT(1) STATIC INIT('1'B);
DCL $ERROR FIXED BIN(15,0) INIT(0);
DCL $NOT_DONE(20) BIT(1);
DCL $TMP_VAL FLOAT BIN;
DCL ($RD_LP$,$R_L) LABEL;
DECLARE
   1 DESIGN,
    2 OUTR,
     3 R1 PIC'zzz9v.99',
     3 R3 PIC'zzz9v.99',
     3 I1 PIC'zzz9v.99',
     3 I2 PIC'zzz9v.99',
     3 I3 PIC'zzz9v.99',
     3 W PIC'zzz9v.99';
DECLARE
   1 PARAM,
    2 INR,
     3 V1 PIC'zzz9v.99',
     3 V2 PIC'zzz9v.99',
     3 V3 PIC'zzz9v.99';
DECLARE
   1 INTERIM,
```

Figure 4.5 (c) Continued

```
    2 $D$1 BIT(1),
    2 $D$2 BIT(1),
    2 $D$3 BIT(1),
    2 $D$4 BIT(1),
    2 $D$5 BIT(1);
DCL (TRUE,SELECTED) BIT(1) INIT('1'B);
DCL (FALSE,NOT_SELE,NOT_SELECTED) BIT(1) INIT('0'B);
ON ENDFILE(PARAMS) BEGIN;
ENDFILE$PARAMS='1'B;
 $FB49=COPY(' ',21);
 END;
ON UNDEFINEDFILE(ERRORF)  ERRORF_BIT='0'B;
DECLARE PLI$_CNVERR GLOBALREF VALUE FIXED BIN(31);
DECLARE RMS$_RLK GLOBALREF VALUE FIXED BIN(31);
ON ERROR BEGIN;
IF ONCODE()=RMS$_RLK THEN GOTO $RD_LP$;
IF $ERROR=0 THEN CALL RESIGNAL();
IF ONCODE()=PLI$_CNVERR THEN DO;
  $ERROR=1;
  IF ERRORF_BIT & $ERROR>0 THEN WRITE FILE(ERRORF) FROM ($ERROR_BUF);
 END;
  ELSE CALL RESIGNAL();
END;
OPEN FILE(PARAMS) INPUT SEQL RECORD;
$ERROR=1;
READ FILE(PARAMS) INTO (PARAM);
  IF $ERROR=0 THEN $ERROR=1;
$ERROR_BUF=$RV50;
$no_of_ass=4;
$no_of_var=2;
begin;
dcl $coeff($no_of_ass+1,$no_of_var+1) float bin(53);
dcl $operation($no_of_ass+1) fixed bin(31,0);
dcl $result($no_of_ass) float bin(53);
dcl ($iii,$iiii,$icase) fixed bin(31,0);
do $iii=1 to $no_of_ass+1;
  $operation($iii)=0;
  do $iiii=1 to $no_of_var+1;
    $coeff($iii,$iiii)=0;
  end;
end;
  $coeff(1,2)=$coeff(1,2)+PARAM.V2;
  $coeff(1,3)=$coeff(1,3)+PARAM.V3;
  $operation(2)=-1;
  $coeff(2,2)=$coeff(2,2)-1;
  $coeff(2,3)=$coeff(2,3)-1;
  $coeff(2,1)=$coeff(2,1)+6;
  $operation(3)=-1;
  $coeff(3,3)=$coeff(3,3)-1;
  $coeff(3,1)=$coeff(3,1)+4;
  $operation(4)=-1;
  $coeff(4,2)=$coeff(4,2)-1;
  $coeff(4,1)=$coeff(4,1)+4;
  call simplex($coeff,$no_of_ass,$no_of_var,$operation,$icase,$result)
;
```

Figure 4.5 (c) Continued

```
  DESIGN.I2=$result(1);
  DESIGN.I3=$result(2);
  DESIGN.W=$result(3);
end;
/* 8 */DESIGN.I1=DESIGN.I2+DESIGN.I3;
/* 6 */DESIGN.R1=(PARAM.V1-PARAM.V2)/DESIGN.I1;
/* 7 */DESIGN.R3=((PARAM.V1-PARAM.V3)-DESIGN.I1*DESIGN.R1)/DESIGN.I3;
WRITE FILE(DESIGNT) FROM ($RV42);
CLOSE FILE(DESIGNT);
RETURN;
END CIRCUIT;
```

Question 3:

To solve question 3, find the formula for the ratio cost/power first.

```
cost = k2*V1*I1+k3*(V1*I1-W)
     = (k2+k3)*V1*I1-K3*W
cost/power = (k2+k3)*V*I1/(k1*W)-k3/k1;
```

Using constants c1=(k2+k3)/K1, c2=k3/k1; the above formulas can be written as:

```
cost/power = c1*V1*I1/W -c2;
```

Since the purpose is to adjust the currents I2 and I3 to make cost/power maximal, the values of I2 and I3 do not depend on c1 and c2. We can drop out c1 and c2 to make the problem simple. The final optimization problem can be written as (the constraints are added arbitrarily to make the results meaningful.)

```
mu = maximize(V1*(I2+I3)/(V2*I2+V3*I3)) DEC_VAR: I2, I3;
I2,I3>=0.5;
I2<=6;
I3<=4;
```

If we put these assertions together with the two equations in the solution for question 1, we get the specification for question 3.

Figure 4.6 (a) The user specification for Question 3

```
module: circuit;
source: param;
target: design;

1 param is file,
  2 inr is rec,
    3 (v,v1,v2) are field(pic 'zzz9v.99');

1 design is file,
  2 outr is rec,
    3 (r1,r2,i1,i2,i3,w) are field(pic 'zzz9v.99');

i1*r1=v1-v2;
i1*r1+i3*r3=v1-v3;
i1=i2+i3;

w=minimize(v*(i2+i3)/(v2*i2+v3*i3)) variable i2,i3;
i2,i3>=0.5;
i2<=4;
i3<=4;
i2+i3<=6;
```

Figure 4.6 (b) The flowchart report for Question 3

```
                                  *****  FLOWCHART REPORT  *****

                          NEST
NAME                      LVL: DESCRIPTION                          EVENT
----                      ---- -----------                          -----

CIRCUIT                        MODULE NAME                         PROCEDURE HEADING
PARAM                          FILE                                OPEN FILE
INR                            RECORD IN FILE PARAM                READ RECORD
V1                             FIELD IN RECORD INR
V2                             FIELD IN RECORD INR
V3                             FIELD IN RECORD INR
                             1 METHOD: SIMPLEX ($MAINBLOCK         MATHEMATICAL PROGRAMMING BLOCK
I2                             FIELD IN RECORD OUTR
I3                             FIELD IN RECORD OUTR
AAS89                          EQUATION
MU                             FIELD IN RECORD OUTR                TARGET OF EQUATION: AAS89
AAS813                         INEQUALITY CONSTRAINT
$D$5                           FIELD
AAS812                         INEQUALITY CONSTRAINT
$D$4                           FIELD
AAS810                         INEQUALITY CONSTRAINT
$D$1                           FIELD
AAS811                         INEQUALITY CONSTRAINT
$D$3                           FIELD
AAS810AB                       INEQUALITY CONSTRAINT
$D$2                           FIELD
                             1 END METHOD: SIMPLEX ($MAINBLOCK     MATHEMATICAL PROGRAMMING BLOCK
AAS88                          EQUATION
I1                             FIELD IN RECORD OUTR                TARGET OF EQUATION: AAS88
AAS86                          EQUATION
R1                             FIELD IN RECORD OUTR                TARGET OF EQUATION: AAS86
AAS87                          EQUATION
R3                             FIELD IN RECORD OUTR                TARGET OF EQUATION: AAS87
AAS814                         EQUATION
W                              FIELD IN RECORD OUTR                TARGET OF EQUATION: AAS814
OUTR                           RECORD IN FILE DESIGN               WRITE RECORD
DESIGN                         FILE                                CLOSE FILE
                               END
```

Figure 4.6 (c) The generated program for Question 3

```
CIRCUIT: PROCEDURE OPTIONS(MAIN);
DCL $MALSTR CHAR(1);
DCL PARAMS RECORD SEQL INPUT;
DCL $FSTPARAMS BIT(1) INIT('1'B);
DCL ENDFILE$PARAMS BIT(1) INIT('0'B);
DCL $FB51 CHAR(21) VARYING INIT('');
DCL $FX51 FIXED BIN(31,0);
DCL $RV52 CHAR(21) BASED(ADDR(PARAM)) ;
dcl $no_of_var fixed bin(31,0);
dcl search entry((*)float bin(53),float bin(53),fixed bin(31,0)
,entry,entry,fixed bin(31,0),(*)float bin(53),bit(1));
$evaluate1: proc($xxx$) returns(float bin(53));
dcl $xxx$(*) float bin(53);
return((PARAM.V1 *($xxx$(1) +$xxx$(2) )/(PARAM.V2 *$xxx$(1) +PARAM.V3 *$xxx$(2)
))**2);
end;
$inside1: proc($xxx$) returns(bit(1));
dcl $xxx$(*) float bin(53);
  $D$5 =$xxx$(1) +$xxx$(2) <=6 ;
  $D$4 =$xxx$(2) <=4 ;
  $D$1 =$xxx$(2) >=0.5 ;
  $D$3 =$xxx$(1) <=4 ;
  $D$2 =$xxx$(1) >=0.5 ;
dcl ( $D$5 ,$D$4 ,$D$1 ,$D$3 ,$D$2 ) bit(1);

return( $D$5 &$D$4 &$D$1 &$D$3 &$D$2 );
end;
DCL $RV43 CHAR(49) BASED(ADDR(DESIGN)) ;
DCL DESIGNT RECORD SEQL OUTPUT ENV(MAXIMUM_RECORD_SIZE (49));
DCL $FSTDESIGNT BIT(1) INIT('1'B);
OPEN FILE(DESIGNT) OUTPUT;
DCL $ERROR_BUF CHAR(270) VAR;
DCL ERRORF FILE RECORD OUTPUT;
DCL ERRORF_BIT BIT(1) STATIC INIT('1'B);
DCL $ERROR FIXED BIN(15,0) INIT(0);
DCL $NOT_DONE(20) BIT(1);
DCL $TMP_VAL FLOAT BIN;
DCL ($RD_LP$,$R_L) LABEL;
DECLARE
   1 DESIGN,
    2 OUTR,
     3 R1 PIC'zzz9v.99',
     3 R3 PIC'zzz9v.99',
     3 I1 PIC'zzz9v.99',
     3 I2 PIC'zzz9v.99',
     3 I3 PIC'zzz9v.99',
     3 W PIC'zzz9v.99',
     3 MU PIC'zzz9v.99';
DECLARE
   1 PARAM,
    2 INR,
     3 V1 PIC'zzz9v.99',
     3 V2 PIC'zzz9v.99',
     3 V3 PIC'zzz9v.99';
```

Figure 4.6 (c) Continued

```
DECLARE
   1 INTERIM,
    2 $D$1 BIT(1),
    2 $D$2 BIT(1),
    2 $D$3 BIT(1),
    2 $D$4 BIT(1),
    2 $D$5 BIT(1);
DCL (TRUE,SELECTED) BIT(1) INIT('1'B);
DCL (FALSE,NOT_SELE,NOT_SELECTED) BIT(1) INIT('0'B);
ON ENDFILE(PARAMS) BEGIN;
ENDFILE$PARAMS='1'B;
 $FB51=COPY(' ',21);
 END;
ON UNDEFINEDFILE(ERRORF)  ERRORF_BIT='0'B;
DECLARE PLI$_CNVERR GLOBALREF VALUE FIXED BIN(31);
DECLARE RMS$_RLK GLOBALREF VALUE FIXED BIN(31);
ON ERROR BEGIN;
IF ONCODE()=RMS$_RLK THEN GOTO $RD_LP$;
IF $ERROR=0 THEN CALL RESIGNAL();
IF ONCODE()=PLI$_CNVERR THEN DO;
  $ERROR=1;
  IF ERRORF_BIT & $ERROR>0 THEN WRITE FILE(ERRORF) FROM ($ERROR_BUF);
 END;
  ELSE CALL RESIGNAL();
END;
OPEN FILE(PARAMS) INPUT SEQL RECORD;
$ERROR=1;
READ FILE(PARAMS) INTO (PARAM);
  IF $ERROR=0 THEN $ERROR=1;
$ERROR_BUF=$RV52;
$no_of_var=2;
begin;
dcl $initial($no_of_var) float bin(53);
dcl ($result($no_of_var),$eps) float bin(53);
dcl ($iii,$times) fixed bin(31,0);
do $iii=1 to $no_of_var;
  $initial($iii)=1;
end;
$times=300;
$eps=0.01;
  call search($initial,$eps,$times,$evaluate1,$inside1,$no_of_var,
$result,'1'b);
  DESIGN.I2=$result(1);
  DESIGN.I3=$result(2);
  DESIGN.MU=$evaluate1($result);
end;
/* 8 */DESIGN.I1=DESIGN.I2+DESIGN.I3;
/* 6 */DESIGN.R1=(PARAM.V1-PARAM.V2)/DESIGN.I1;
/* 7 */DESIGN.R3=((PARAM.V1-PARAM.V3)-DESIGN.I1*DESIGN.R1)/DESIGN.I3;
/* 14 */DESIGN.W=PARAM.V2*DESIGN.I2+PARAM.V3*DESIGN.I3;
WRITE FILE(DESIGNT) FROM ($RV43);
CLOSE FILE(DESIGNT);
RETURN;
END CIRCUIT;
```

4.3.1. Example of Use of MATHMODEL In Large Scale Mathematical Modelling

An important advantage in using MATHMODEL in development of large scale mathematical models is the capability to generate individual model components which can be plugged in the larger model. Large scale mathematical models are generally constructed over a prolonged period with numerous participants acting semi-independently. Use of MATHMODEL can fit into this mode of development. MATHMODEL generates programs for respective procedures or tasks used in the large scale model. Demonstrating this capability is beyond the scope of this project. However, we cite the following relevant experience with use of CCCC's MODEL in a separate project. This consisted of activity sponsored by the Naval Surface Weapon Center (NSWC) as part of DoD STARS project. This, so called "shadow" project, consisted of repeating the development of a large signal processing system illustrated in Figure 4.7. The requirement was to develop procedures to be plugged in for the Multiping Processing Function shown centrally in Figure 4.7. Three parallel efforts were conducted and the labor invested was recorded. The three efforts used different development tools: GE used manual development, NSWC used a Computer Aided Software Engineering (CASE) system called EPOS and CCCC used MODEL. The comparative labor costs are shown in Figure 4.8. Use of MODEL has a significant advantage by a ratio of nearly 3:1 over the other modes of development.

Figure 4.7: Overview of Signal Processing System With MULTIPING Component

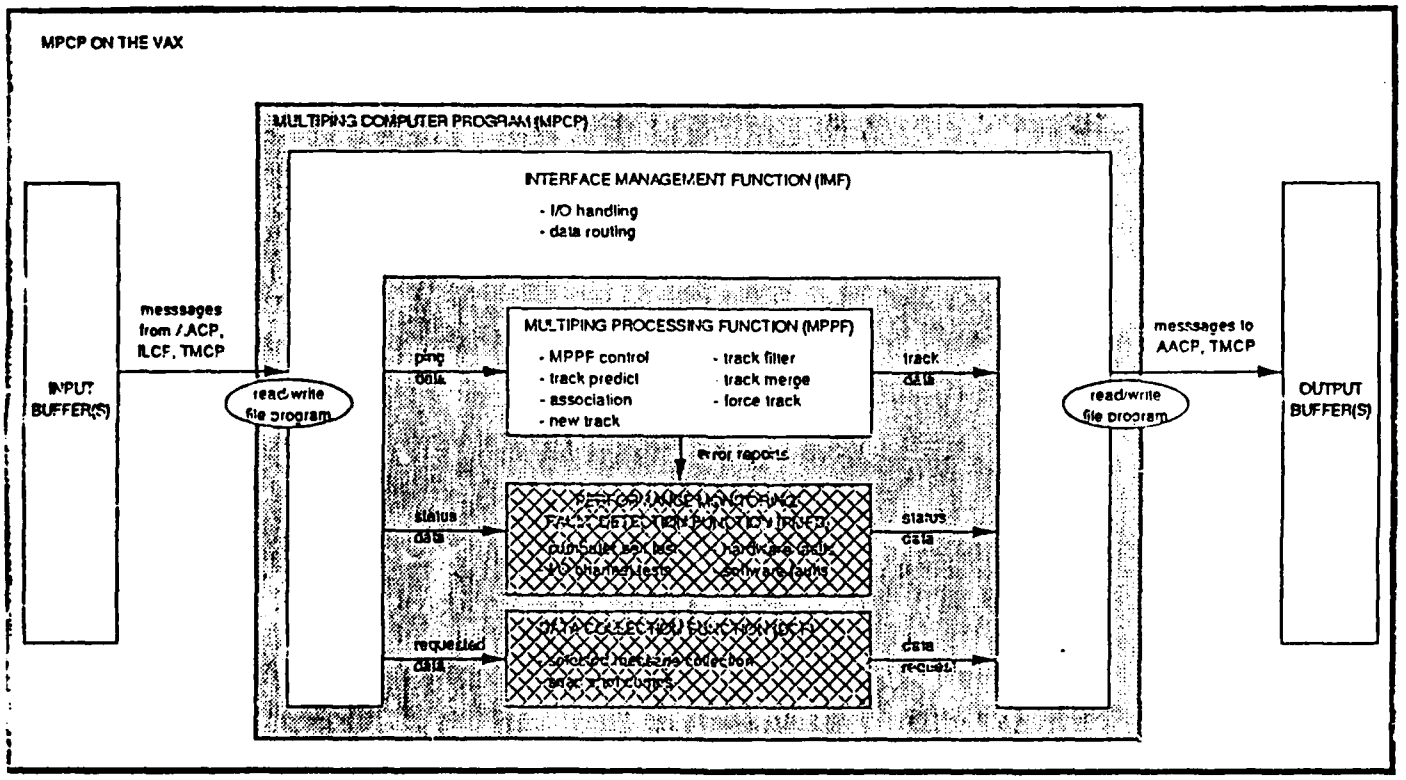

Figure 4.8: Results of Evaluation of the MULTIPING Shadow Projects

Stars Shadow Project Metrics

TOTAL LABOR

(includes Administration, Requirements, Detail Design, Testing and Integration)

	Man Days	Ada SLOC	SLOC/Man Day
GE	1440	3400	2.36
NSWC	578	2100	3.63
CCCC	155	1400	9.03

Extrapolation of Labor and SLOC to Implement Entire System

	Equivalent Implemented CMS LOC	% Implemented	Actual Labor	Projected Labor	Projected LOC
GE	7400	100	1440	1440	3400
NSWC	5000	68	578	850	3100
CCCC	2100	28	155	553	5000

All Data Taken from NSWC 3/6/89 Presentation

5. TASK-3: INVESTIGATION OF THE MARKETPLACE FOR MATHMODEL

The marketplace for MATHMODEL is wide and varied. In the following we consider only the initial product and the respective marketplace that will be addressed immediately in Phase III. There is a potential to expand the initial product and marketplace greatly.

5.1. Definition of the Initial MATHMODEL Product

The initial MATHMODEL system that will be offered to users in Phase III is envisaged to consist of the following:

MATHMODEL as described in Section 2.

Platform: operating system.	Digital VAXstation with large color screen. VMS
Languages:	PL/1, Ada, C and Fortran
Price:	$50,000 - $100,000 Includes $20,000 - $40,000 for a) workstation b) software from cooperating vendors for graphics, documentation and library.

5.2. Definition Of The Initial Marketplace For The MATHMODEL Product

We divided the mathematical modelling users into three classes:

1. Those being trained in applied mathematics: This is a very large market that includes college students. There are a number of systems in existence and being developed for this market.
2. Mathematicians seeking analytical insight into problems in mathematics. This class of users requires sophisticated symbolic manipulation. They have been using MACSYMA [Moses '71]
3. Large scale model developers drawn from a US community of 1 million engineers, physical scientists, economists, business specialists and other social scientists. Nearly 10%, of this community, or 100,000 developers use mathematical models for forecasting, simulation and complex operational decisions.

MATHMODEL is oriented to the last class, and particularly to large scale model developers.

The initial market size is further constrained as follows:

Application Areas:

- Mathematical modelling in engineering and physical

sciences.

- Simulation systems.
- Training systems.

Our investigation showed that if we limit the application areas to only mathematical modelling then the economic justification for the product is marginal. It will therefore be necessary to cover the closely related application areas of simulation systems and training systems.

The overall user community in the US is estimated from census data:

Analysts	$150,000
Engineers	$750,000
Physical Scientists	$200,000
	$1,100,000

Only 10% of these are directly active in the above application areas. Considering, that the MATHMODEL product will be offered in a personal workstation:

The marketsize is in the order of 10,000 units.

The total value will be in $.5 - 1.0 billion.

Of this over 60% will be attributed to MATHMODEL and the remainder to the workstation and associated software

5.3. Marketing Strategy

Introducing new technology has historically been a slow process. The company is currently introducing the MODEL program generation system into the large software project marketplace. A similar strategy can be followed in offering MATHMODEL. MATHMODEL is a complex product. It requires direct selling. Early establishment of the high quality of a product is critical to successful market penetration. It requires first rate technical support in terms of documentation, training and consulting.

It will be necessary to address very specific market segments in order to maximize the effectiveness of resources in selling the product. The target clients should have all of the five attributes:

1. Aerospace, defense, engineering organizations;

2. Large scale mathematical modelling, simulation and trainer development projects;
3. Projects currently funded, and in the early stages of development;
4. Currently utilizing VAX or IBM hardware;
5. Currently using CAD/CAM and/or CASE products.

Initial marketing will be oriented to proving the advanges of MATHMODEL. This can be accomplished by selling a *Transfer of Technology* package which derives revenue during the customers' evalution period. This package will include formalized training sessions and consulting assistance, in order to convince clients of MATHMODEL's capabilities within their environment.

The marketing channels will be as follows:

1. Direct selling activities: Most of the selling will be done with direct salesmen accessing both the prime and sub-contractors on large scale projects.
2. Indirect sales through distributors: This will be directed mainly to foreign markets. We have a distribution agreement with a German organization which provides support to the client. Training will be conducted by CCCC in all aspects of the product technology.
3. Outside cooperation in selling activities:
    a. In conjunction with hardware manufacturers: IBM and Digital have been particularly supportive in this area, providing leads and setting up seminars. MATHMODEL will be presented at sales meetings, leading to joint sales calls with IBM's and Digital's representatives.
    b. Alliance with Software Vendors: With the completion of the interfaces, opportunities will arise that will enhance our selling effort.

5.4. Competitive Mathematical Modelling Systems

The current mathematical modeling systems have been developed for specific classes of users and applications. For this reason, most of them incorporate one or a few solution methods needed for the respective applications. Also their input languages and organizations are narrowly oriented to the respective solution method used. They are typically closed systems, difficult to expand capabilities. As the application areas change and the respective mathematical models increase in size and detail, changes in the mathematical modeling system become mandatory. The history of these systems is that of continuous modification which require major redesign at great expense.

The reviewed mathematical modeling systems illustrate the above points.

TROLL [Troll 76] is oriented to regression, data analysis, and solution of linear and nonlinear equations. TROLL is open-ended in the sense of adding solution methods for regression and simultaneous equations. However, a major change would be required to incorporate other approaches to solutions, such as optimizations. The input language is specific to the above solution approaches and symbolic manipulation is not provided. There have been major investments in improving efficiency, especially due to the interpretive methods used. The original capabilities have proved to be inadequate and the system evolved in a major way.

LINDO [Schrag 86] is a linear, integer, and quadratic programming solver. It is convenient for a user to type in small problems interactively. It does not support simultaneous equations. The user can compose or use a library of Fortran subroutines to organize data or produce reports. This capability makes it versatile but difficult to use. It is not an open-end system and would be very difficult to add radically different solution methods.

GINO [Lieman 86] is an interactive optimizer implemented on IBM PC. It is a successor of LINDO for solving nonlinear programming problems. The Generalized Reduced Gradient algorithm is built-in. A user has to provide Fortran subroutines to evaluate the objective and constraint functions. The gradient algorithm requires specification of partial derivatives. They can be provided optionally by the user through Fortran subroutines, or the derivatives can be estimated by the system numerically. The system does not support simultaneous equations. It is not an open-ended system and would be very difficult to add new solution methods.

EMP [Schitt 88] is an interactive system that supports model building, numerical solution and data processing of mathematical programming (both linear and nonlinear) problems. Linear constraints are entered by specifying constraint coefficients in an array. All other aspects are defined by user-provided Fortran statements. The system helps a user select solution methods. It is an open system with many built-in solution methods.

GAMS [Meerau 83] is a widely used system. It solves linear and nonlinear optimization problems. It has the capability to accept the modeler's form of the problem as input and translates it into the form required by the algorithms. For nonlinear programming problems, the system computes the first derivatives numerically.

MINOS [Waren 87] is specially built for large sparse nonlinear programming. A user has to provide Fortran subroutines to calculate the nonlinear constraints. MINOS is currently the most

widely used large scale nonlinear programming solver. It requires that the variables be sorted by the user so the nonlinear ones appear first. The user-provided Fortran subroutines must compute only those parts of the nonlinear constraints that depend on the nonlinear variables. This restriction makes MINOS difficult to use. MINOS has been merged into GAMS as a nonlinear problems solver. It is based on the Generalized Reduced Gradient method.

GXMP [Dolk 86] is an modeling system for linear programming. The system accepts mathematical formulas as input. It incorporates a symbolic manipulation which transforms a user's form into a form required by solution method. It is an open system with many solution methods.

TSP [Drud 83] is a time series processor which has been used widely since the late 1960s. It is possible to redefine the endogenous and exogenous variables without rewriting the equations of the model. TSP has a primitive capability to handle symbolic expressions, but a user has to know the internal structure to use this capability. It lacks solution of optimization problems. The system suffers from low efficiency primarily due to use of a command language.

STS-SYSTEM [Schlei 80] is used for system simulation with special emphasis on econometric methods. It is the integration of database management, statistical parameter estimation, and documentation by a simple command language. The command language makes it a prescriptive approach. It incorporates analysis for reordering and renormalization of equations. STS-SYSTEM utilizes symbolic manipulation.

CAMP [Sagie] offers a simple and coherent tool for planning. It covers different facets of the planning activity: data management, linear programming, statistical analysis, graphics, and word processing.

6. CONCLUSION

6.1. Accomplishments and Plans

MATHMODEL has the potential for becoming the next generation of mathematical modelling systems and providing an order-of-magnitude improvement over current methods. Mathematical modelling is at the core of technical developments and planning. An improved mathematical modelling system has the potential of producing better products and systems and

improving overall US competitiveness.

In Phase I, we overcame the problems of reliability and documentation, common in research projects, and we also added essential operations on entire variable structures (arrays, files, etc.). This was achieved by combining the original University of Pennsylvania version of MATHMODEL with CCCC's MODEL. We also investigated demonstrating MATHMODEL in ways that would convince potential users of its advantages.

However, to be accepted widely, it is necessary in Phase II to make MATHMODEL much more attractive to its community of users, as follows.

Users have shown considerable preference for using a personal workstation. We propose to concentrate initially on Digital's VAX workstation, which is widely used. Digital is providing VAXstations with increasingly higher speeds and with vectorizer attachments. The VAXstation has a mature software foundation.

Next, users insist on a man/machine interface that reflects a systematic methodology and discipline. The essense of mathematical modelling may be expressed by the words *prototyping* and *reusability*. Namely, a model is expanded progressively from a core to more detailed sections. Each step involves formulation and testing in a trial and error process. Each step reuses external models as well as the previously developed components. It is necessary to develop the graphics and databases to support this methodology. This will require an innovative approach to specifying models. It will use recent advances in graphics and Computer Aided Software Engineering (CASE).

At the end of Phase II, we can have a technically complete a commercial product. We will still need, to provide in Phase III, commercial level training, documentation and marketing and to conduct beta testing. There is enormous interest in automation of mathematical modelling and we expect an abundance of offers to cooperate with us in Phase III in the initial marketing introduction.

The opportunity is then to provide an order-of-magnitude improved next generation mathematical modelling system. Mathematical modelling is at the core of new technical developments and ecomonic planning. We are aware of very large systems that are becoming obsolete and for which there is scant documentation. The Department of Defense, for example,

has these problems. Using MATHMODEL will reduce the required expert resources, which are scarce, and will cut the large costs and long development terms. An improved mathematical modelling system has the potential of producing better products and systems and improving overall US competitiveness.

CCCC is continuing to enhance the MODEL system and these enhancements will be available to MATHMODEL as well, at no additional direct investment. Of particular relevance to MATHMODEL are the following capabilities:

- To generate the programs in Fortran
- To generate programs which use vectorizers to speed up evaluation of mathematical models.

6.2. Prototyping and Reusability Development Mode

This mode of development is naturally iterative. Sometimes it is referred to as *spiral*, referring to repetition of a sequence of steps as the model grows. It can be contrasted with the so-called *waterfall* mode where the development goes through a linear sequence of phases. The portability and reusability mode is illustrated in Figure 6.1. Starting with a mathematical modelling requirement, one can select for first attack any part of the model. The core of the model can be the most difficult or the central requirement. It can then be expanded progressively to include the less critical or less central parts. There is no need for a user to observe sequential order of events in the model. Starting with the core model, the development consists of conceptualizing the key objects, namely the variables, and composing respective equations. MATHMODEL is then used at each step to check the logic and generate programs. The core can then be tested independently. If necessary, MATHMODEL can also be used to generate test data. In subsequent steps, objects and equations are added progressively. MATHMODEL generates new programs for each step that synthesize all the previous steps. At first, the user seeks only model correctness, with little regard for elegance of representations and efficiency of computations. Once the user is satisfied with the results of the test, he/she can review the model to improve it's representation and efficiency. New programs are then generated. Finally, this portion of a model can be "plugged-in" to operate with the rest of the larger model.

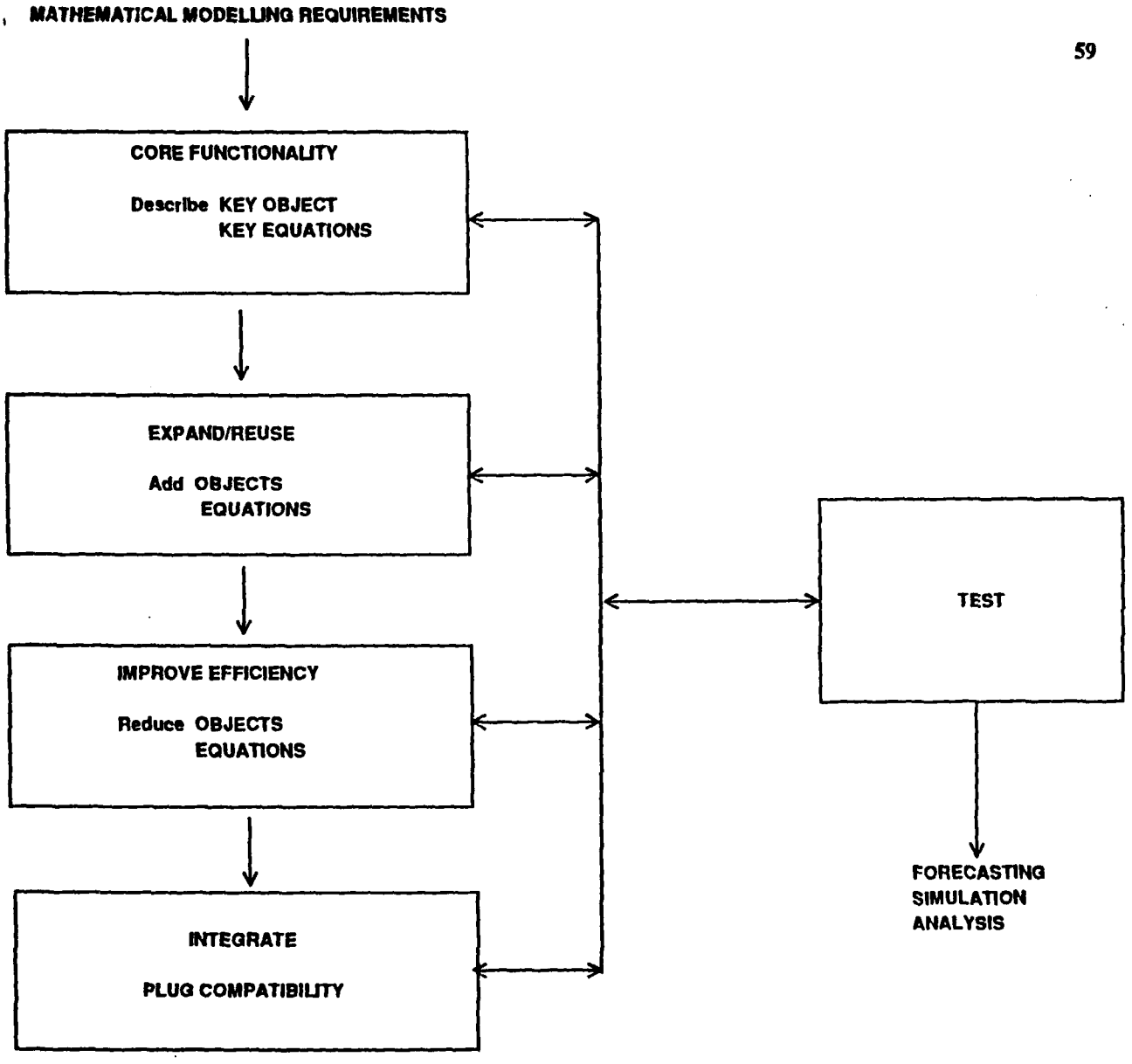

Figure 6.1: Prototyping and Reusability Process.

The problem of reusing fragments of mathematical models is very similar to the problem of *reusability* of software, which has been widely researched. In the following, we borrow existing concepts and modify them for mathematical modelling.

Mathematical model reusability is the underlying technique for two types of development activities. First, implementation of a new model would involve selection of building blocks from a "warehouse" of existing model fragments. Second, enhancing an existing model, such as increasing the throughput or expanding the scope, would involve adding or modifying individual "plug-ins", similiar to expanding a hardware system.

We refer to the first mode as "modelling-in-the large", where the plug-in is an entire task or procedure in the total model programs. The second mode is "modelling-in-the-small", where fragments of models are combined to create a procedure or task for an expanded model.

Both modes have been demonstrated in Phase I and reported in the Final Report. Our reusability approach is based on retaining in the "warehouse" (database) blocks of equations. Each equation expresses a rule applied to the model's variables. The language of equations is general purpose and can be used to express the concept embedded in any model fragment. Each equation is a self-contained rule and produces no "side effects" as in procedural programs. Therefore, understanding equations is much easier than understanding an equivalent program. Equations may be in any arbritary order. We can use MATHMODEL to automatically integrate the selected equations blocks into a corresponding procedural program. MATHMODEL can translate the equation blocks into an integrated program. The generated programs are efficient, competitive with manually developed equivalent programs. In this way, MATHMODEL solves the problem of fitting together blocks into an effective integral entity.

There are other advantages. MATHMODEL generates declarations of interim variables that provide the precision required in the output. A mathematical model includes declarations of input and output. The precision of interim variables is derived from the declarations of output. MATHMODEL also performs logical checking of the submitted equations and declarations. When it discovers an incompleteness or inconsistency, it either makes a correction or it instructs the user on how to make the necessary correction. The user may have to compose some custom equations not found in the "warehouse" in order to make the indicated correction. MATHMODEL generates 100% of the procedural program and there is no need for the user to

make any additions or modifications to the code.

In order to apply this concept of reusability, it is necessary to create a "warehouse" of blocks of equations for old models. For instance, a signal analysis "warehouse" would include equation blocks for different algorithms for Fast Fourier Transforms, noise discrimination and analysis of signals. The blocks may be structured as "parts", that fit into "assemblies", etc.

The developer of a mathematical model will draw a block diagram on the screen of a terminal, similiar to those used in hardware systems, showing the new architecture of the mathematical model. The boxes in this diagram must be related to the respective equation blocks. The connecting lines in this diagram must be related to the variables that flow between the respective blocks. The diagram would be drawn with the aid of a graphics workstation

To construct a system or to enhance its functionality, the block diagram is used to select automatically the blocks of equations from the "warehouse". The blocks of equations, together with declarations of inputs and outputs, are then submitted to MATHMODEL to generate the desired programs. Any desired modifications are performed by modifying, adding or deleting equations and then repeating the generation of a new programs that reflect the changes.

1. REFERENCES

1. T. Agerwala, Arvind, "Data Flow Systems," *Computer*, February, 1982.

2. J.R. Allen and K. Kennedy, "Automatic Loop Interchange," *Proc. of the ACM SIGPLAN Symposium on Compiler Construction*, SIGPLAN Notices V19 #6, June 1984.

3. E. Ashcroft, Z. Manna, "The Translation of Goto Programs to While Programs", *Proceedings, IFIP Congress 1971*, North-Holland Publ. Co. Amsterdam, pp. 250-255, 1972.

4. E. Ashcroft and W.W. Wadge, "Lucid, A Nonprocedural Language with Iteration," *Communications of the ACM*, V20 #7, July 1977.

5. J. Backus, "Can Programming be Liberated From the Von Neumann Style? A Functional Style and its Algebra of Programs," *Communications of the ACM*, V21 #8, August, 1978.

6. B.S. Baker, "An Algorithm For Structuring Flowgraphs," *Journal of the ACM*, V24 #1, January 1977.

7. R. Balzer, "Transformational Implementation: An Example," *IEEE Transactions on Software Engineering*, V7 #1, January 1981.

8. J Baron, B. Szymanski, E. Lock and N. Prywes, "An Argument for Nonprocedural Languages," *Proc. Workshop Role of Languages in Problem Solving-1*, 1985.

9. S. Basu and J. Misra, "Proving Loop Programs," *IEEE Transactions on Software Engineering*, V1 #1, March 1975.

10. R.K. Boxer, "A Translator From Structured FORTRAN to Jovial/J73," *Proc. of the IEEE National Aerospace and Electronics Conference (NAECON-83)*, 1983.

11. J.M. Boyle and M.N. Muralidharan, "Program Reusability Through Program Transformation," *IEEE Transactions on Software Engineering*, V10 #5, September 1984.

12. E. Bush, "The Automatic Restructuring of Cobol," *Proc. of the IEEE Conf. on Software Maintenance*, November 1985.

13. T. Cheng, E. Lock and N. Prywes, "Use of Very High Level Languages and Program Generation by Management Professionals," *IEEE Transactions on Software Engineering*, V10 #5, September 1984.

14. Computer Command and Control Company, "The MODEL language Usage and Reference Guide -- Non-Procedural Programming for Non-Programmers," 2401 Walnut, Philadelphia, PA 19103, 1987.

15. Digital Equipment Corporation, "VAX-11 FORTRAN User's Guide," *Software Distribution Center, Digital Equipment Corporation*, Maynard, MA 01754, 1979.

16. C.G. Faust, "Semiautomatic Translation of Cobol into Hibol," *(MS Thesis) MIT/LCS/TR-256*, March 1981.

17. R.W. Floyd, "Assigning Meaning to Programs," in *Proc. Symp. Applied Math.*, vol. 19, pp. 19-32, 1967.

18. M.J.C. Gordon, "The Denotational Description of Programming Languages, An Introduction," Springer-Verlag, 1979.

19. C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," *Communications of the ACM*, V12 #10, October 1969.

20. C.R. Hollander, "Decompilation of Object Programs," *Stanford Electronics Lab TR 54*, January 1973.

21. G.L. Hopwood, "Decompilation", *Ph.D. Thesis*, University of California, Irvine, 1978.

22. B.C. Housel III, "A Study of Decompiling Machine Languages Into High-Level Machine Independent Languages", *Ph.D. Thesis*, Purdue University, August 1973.

23. B.C. Housel and M.H. Halstead, "A Methodology for Machine Language Decompilation," *IBM Research Report RJ1316 (No. 20557)*, 17 pages, December 6, 1973.

24. S. Katz and Z. Manna, "Logical Analysis of Programs," *Communications of the ACM*, V19 #4, April 1976.

25. D.J. Kuck, R.H. Kuhn, D.A. Padua, B. Leasure and M. Wolfe, "Dependence Graphs and Compiler Optimizations," *Proc. 8th ACM Symp.* pp.207-218, 1981.

26. K.S. Lu, "Program Optimization Based on a Non-Procedural specification," Ph.D. dissertation, Department of Computer Science, University of Pennsylvania, 1981.

27. Z. Manna, "Mathematical Theory of Computation," McGraw Hill Book Company, 1974.

28. J. McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation By Machine," *Communications of the ACM*, V3 #4, April, 1960.

29. J.R. McGraw, "The VAL Language: Description and Analysis," *ACM Transactions on Programming Languages and Systems*, V4 #1, January, 1982.

30. M.J. O'Donnell, "Equational Logic as a Programming Language," The MIT Press, 1985.

31. H. Partsch, R. Steinbruggen, "Program Transformation Systems," *Computing Surveys*, V15 #3, September, 1983.

32. L. Paulson, "Compiler Generation from Denotational Semantics," in *Methods and Tools for Compiler Construction*, ed. by B. Lorho, pp. 219-250, 1984.

33. K.M. Pitman, "A FORTRAN to Lisp Translator," *Proc. of the 1979 Macsyma Users' Conference*, June 1979.

34. U. Pleban, "Compiler Prototyping Using Formal Semantics," SIGPLAN Notices, V19 #6, Montreal, June, 1984.

35. N. Prywes and A. Pnueli, "Compilation of Nonprocedural Specifications into Computer Programs," *IEEE Transactions on Software engineering*, V9 #3, May 1983.

36. C. Rich, H.E. Shrobe, R.C. Waters, G.J. Sussman and C.E. Hewitt, "Programming Viewed as an Engineering Activity", MIT, Cambridge, MA, MIT/AIM-459, January 1978.

37. J. Samet, "Experience with Software Conversion," *Software -- Practice and Experience*, V11 #10, 1981.

38. D. Schmidt, "Denotational Semantics - A Methodology for Language Development," Allyn and Bacon, Inc., 1986.

39. M. Shaw and W.A. Wulf, "Abstraction And Verification in ALPHARD: Defining and Specifying Iteration and Generators," *Communications of the ACM*, V20 #8, August 1977.

40. L. Sterling, E. Shapiro, "The Art of Prolog," *The MIT Press*, 1986.

41. B. Szymanski, E. Lock, A. Pnueli and N. Prywes, "On the Scope of Static Checking in Definitional Languages," *Proc. of the ACM Annual Conference*, San Francisco, CA, pp.197-207, October 1984.

42. B. Szymanski, N. Prywes, "Efficient Handling of Data Structures in Definitional Language," *Science of Computer Programming*, pp.221-245 No. 10, October 1988.

43. R.C. Waters, "A System for Understanding Mathematical FORTRAN Programs," MIT, Cambridge, MA, MIT/AIM-368, August 1976.

44. R.C. Waters, "Expressional Loops," *Proc. 10th ACM Symposium Principles of Programming Languages*, pp. 1-10, ACM, 1983.

45. R.C. Waters, "A Method for Analyzing Loop Programs," *IEEE Transactions on Software Engineering*, V11 #11, November 1985.

46. R.C. Waters, "Program Translation Via Abstraction and Reimplementation," *IEEE Transactions on Software Engineering*, V14 #8, August 1988.